Semantic Assumptions and Query Evaluation in Temporal Databases*

Claudio Bettini X. Sean Wang Elisa Bertino Sushil Jajodia

Abstract

When querying a temporal database, a user often makes certain semantic assumptions on stored temporal data. This paper formalizes and studies two types of semantic assumptions: point-based and interval-based. The point-based assumptions include those assumptions that use interpolation methods, while the interval-based assumptions include those that involve different temporal types (time granularities). Each assumption is viewed as a way to derive certain implicit data from the explicit data stored in the database. The database system must use all explicit as well as (possibly infinite) implicit data to answer user queries. This paper introduces a new method to facilitate such query evaluations. A user query is translated into a system query such that the answer of this system query over the explicit data is the same as that of the user query over the explicit *and* the implicit data. The paper gives such a translation procedure and studies the properties (safety in particular) of user queries and system queries.

1 Introduction

A prominent feature of temporal information is the richness in semantics associated with its temporal domains. As an example, consider the temporal relation ACCOUNTS shown in Figure 1. Each tuple of ACCOUNTS corresponds to an account number (AcctNo), account holder (AccHol), account balance (Balance), annual interest rate (AnIntRate), and the time (Time) when the values in the tuple become valid. New entries are added to this relation whenever an event such as opening of an account, a transaction, or a change in interest rates occurs. The values of Time are timestamps consisting of the date (month/day/year) concatenated with the time of the day (up to seconds). When querying ACCOUNTS a user makes some "usual" semantic assumptions on the stored temporal data. For instance, she expects that her bank account balance persists, i.e., the balance stays the same unless another transaction (deposit or withdrawal etc.) is performed. When she asks for the balance at a particular time and no balance amount is stored for that time, she searches for the balance in the last transaction whose timestamp is less than the time in question. Another example of persistence is given by the attribute AccHol whose values persist for each account number.

AcctNo	AccHol	Balance	AnIntRate	Time
1001	J.Smith	1000	3.00	3/3/93:09:01:00
1001	J.Smith	3500	4.00	3/4/93:10:01:55
1500	A.Brady	2000	3.00	3/4/93:11:00:00
2034	T.Ford	500	2.50	3/4/93:12:19:03
1500	A.Brady	1500	3.00	3/4/93:18:00:00
1001	J.Smith	4000	4.00	7/3/93:09:00:00

Figure 1: An instance of the relation ACCOUNTS

Semantic assumptions may also involve different temporal types (time granularities). For example, the attribute AnIntRate can be classified as *liquid*. That is, if its value is 3.00 for a month, we can assume that its value is 3.00 for each second of each day in that month, and at the same time, if its value is 3.00 for each second of each day in a month, we can assume that its value is 3.00 for that month. (The term "liquid" is borrowed from the temporal reasoning community [Sho87].)

The aforementioned semantic assumptions are only some of the assumptions identified by the temporal reasoning community. There are many semantic assumptions that are quite natural in databases but not discussed in that community. For example, the balance at the end of a month is the last balance stored in the relation for that month. In Figure 1, assuming no activities occurred for account 1001 after 7/3/93:09:00:00, the end-of-month balance for July 1993 will be $4,000.

Researchers have long realized such richness of semantics in temporal data [CT85, SS87, Tan87] and provided operators in their query languages to enable users to code semantic assumptions into queries [SS87, Tan87]. Users can use a "last" (or similar) operator to formulate the preceding balance query.

We argue, however, that temporal semantics should be an integral part of temporal databases and the users should not bear the burden of having to incorporate

*The work is supported in part by an ARPA grant, administered by the Office of Naval Research under grant number N0014-92-J-4038. Work of Bettini was performed while visiting George Mason University. Work of Wang is also supported by the NSF grant IRI-9409769. Bettini and Bertino are with the Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy, and Wang and Jajodia with the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030. Emails: {bettini, ebertino}@dsi.unimi.it and {xywang, jajodia}@isse.gmu.edu.

SIGMOD ' 95,San Jose , CA USA
© 1995 ACM 0-89791-731-6/95/0005..$3.50

such semantics in their queries. Users should query the database directly about the balance at a particular time; the system should be able to answer the query appropriately according to the semantics. To have such an ability, the system has to include a precise formalization of temporal semantic assumptions, and evaluate queries according to these assumptions. In this paper, we provide a framework to incorporate temporal semantics into temporal databases and investigate how it can be used to evaluate user queries with embedded semantics.

Each semantic assumption is viewed as a function that obtains implicit information from explicit information. This corresponds to our intuition. Indeed, the persistence of the **Balance** attribute of Figure 1 "generates" for account 1001 the balance amounts equal to \$1,000 for 3/3/93:09:01:01, 3/3/93:09:01:02, 3/3/93:09:01:03, 3/3/93:09:01:04, etc. We denote $\overline{DB}$ as the database DB with all information (implicit and explicit) generated from semantic assumptions.

A user query on a database is viewed as a query on $\overline{DB}$. Indeed, if the query asks for the balance of account 1001 at noon on March 4th, 1993, it should retrieve values from $\overline{DB}$, where the value is \$3,500 for the **Balance** attribute at that time for the account 1001.

However, it is usually impractical for the database to store and manage $\overline{DB}$. Indeed, consider the attribute **AnIntRate** in Figure 1. Since **AnIntRate** is liquid, it is possible to derive values for **AnIntRate** in each possible granularity. There is an enormous effort involved in managing all these derived values. Besides, the user may not be interested in knowing the annual interest rate for all possible time granularities. A similar problem exists for the **Balance** attribute.

Our solution is to translate the user query into a query that incorporates semantic assumptions. For a given user query Q, we need to answer the query Q on $\overline{DB}$, i.e., the database with all semantic assumptions applied. Instead of using semantic assumptions to build $\overline{DB}$, we use them to translate the query Q into another query Q' such that the new query evaluated on the stored database ($Q'[DB]$) is the same as the answer of the user query Q over $\overline{DB}$. In formula, we require that

$$Q'[DB] = Q[\overline{DB}].$$

Obviously, whether there always exists a Q' for each Q largely depends on the languages used. In this paper, we use a single language MQL^F for the semantic assumptions, for user queries Q, as well as for system queries Q'. MQL^F is an extension of the query language introduced in [WJS95] as a general query language on temporal databases. We show that MQL^F is a reasonably powerful language for specifying semantic assumptions. Furthermore, we show that the above Q' always exists in this situation.

Another contribution of the paper is its analysis of safety issues of the temporal database query language MQL^F. Persistence and other semantic assumptions naturally give rise to infinite information. For example, the persistence of the **Balance** attribute dictates that for account 1001 in the **ACCOUNT** relation, there should be one tuple for each second after 3/3/93:09:01:00. Note however that after 7/3/93:09:00:00, all values for each account will stay the same. Such infiniteness, called *eventual uniformity* in this paper, has been accommodated by researchers by using specific data models (e.g., the tuple timestamp $[1, \infty]$ of [Sno84]); however, it has not been analyzed formally in a general setting. In this paper, we study the safety requirement of query languages to yield only eventually uniform results.

In summary, the contribution of the paper is three fold: (1) semantic assumptions are formalized for the purpose of database query evaluation; (2) a method for query evaluation is introduced to incorporate semantic assumptions; and (3) safety issues are analyzed in a general setting.

The rest of the paper is divided into 10 sections. Section 2 briefly reviews related research in the literature. Section 3 defines our temporal data model. In Section 4, we introduce the notion of a general class of semantic assumptions: point-based semantic assumptions. Persistence is a special case of point-based assumptions. Query evaluation with respect to point-based assumptions is discussed in Section 5. Interval-based assumptions that include the liquidity assumption are formalized in Section 6 and query evaluation with respect to such assumptions is studied in Section 7. In Section 8, the above two kinds of assumptions are combined in evaluating queries. Safety issues are discussed in Section 9. Finally, the paper is concluded with some remarks on the results of the paper and on future research directions in Section 10. Due to the lack of space, we refer the reader interested in proofs and more details to [BWBJ95].

2 Related research

As mentioned earlier, some of our terms used for semantic assumptions are borrowed from the temporal reasoning community. In particular, the term *liquidity* is from [Sho87], while the term *persistence* has been extensively used in temporal reasoning for planning (e.g., [AKPT91]). The notion of persistence has also been used in the context of temporal databases in [DM87] to make predictions about unknown facts in the database; however, the emphasis of [DM87] is on consistency maintenance of a logical database containing uncertain temporal information about events. Logical databases and uncertain information are beyond the scope of this paper. Instead, we consider general classes of assumptions including persistence and assumptions involving multiple granularities. Our focus is on relational-like

databases and query evaluation on these databases.

Various semantic assumptions in temporal database settings are perhaps recognized first by Clifford and Warren [CW83]. The earliest systematic study is however performed by Segev and Shoshani [SS87]. They recognized various properties of *time sequences*, such as "stepwise constant" and "continuous", and provided a number of functions to be used in user query languages to accommodate these properties. A larger set of sophisticated operators for the same purpose has been proposed by Tansel in [Tan87]. By using these operators, many semantic assumptions can be coded into user queries. These works differ in two respects from the current one. First, in this paper, we formalize the notion of semantic assumption, while in these earlier studies, no general definition is given. Second, we assume that the user queries the database with the assumptions built in, instead of having to code them into queries. Thus, in this study, the query language for the user is a simple extension of the relational calculus (the extension is for incorporating the temporal dimension, not for semantic assumptions).

Other researchers [CT85, WJL91, CI94] also recognized the importance of semantic assumptions in temporal databases. Clifford [CT85] points out the use of interpolation in temporal databases; however, the query evaluation is not formalized. Clifford and Isakowitz [CI94] deal with formalizing the semantics of *variables* that many temporal data models employ to denote various intuitive semantic assumptions. The work clarifies many vague, although intuitive, notions. However, [CI94] does not address how user queries are evaluated on databases with such variables.

Regarding the safety of temporal queries, the problem is sometimes avoided by assuming a finite time domain and the safety problem reduces to that of standard relational query languages. A recent paper by Clifford, Croker and Tuzhilin [CCT94] mentioned a syntactic safety requirement based on [Ull88] for a logic based temporal query language. However, it is not clear what properties this syntactic restriction leads to.

3 Data Model

The data model used in this paper is based on that introduced in [WJS95, WBBJ94] as a unified interface for accessing different underlying temporal information systems. It is believed that the concepts and the results of this paper are readily translated to other temporal data models.

3.1 Temporal types

We start with defining temporal types that model typical (and atypical) calendar units. We assume there is an underlying notion of absolute time, represented by the set $\mathcal{N}$ of all positive integers.

Definition Let $\mathcal{I}_\mathcal{N}$ be the set of all intervals on $\mathcal{N}$, i.e., $\mathcal{I}_\mathcal{N} = \{[i,j] \mid i,j \in \mathcal{N} \text{ and } i \leq j\} \cup \{[i,\infty] \mid i \in \mathcal{N}\}$.[1] A *temporal type* is a mapping μ from the set of the positive integers (the *time ticks*) to the set $\mathcal{I}_\mathcal{N} \cup \{\emptyset\}$ (i.e., all intervals on $\mathcal{N}$ plus the empty set) such that for each positive integer i, all following conditions are satisfied:

(1) if $\mu(i) = [k,l]$ and $\mu(i+1) = [m,n]$, then $m = l+1$.

(2) $\mu(i) = \emptyset$ implies $\mu(i+1) = \emptyset$.

(3) there exists j such that $\mu(j) = [k,l]$ with $k \leq i \leq l$.

Condition (1) states that the mapping must be *monotonic* and that the intervals corresponding to consecutive ticks should not have a gap between them. Condition (2) disallows a temporal type to map a certain time tick to the empty set unless it maps all subsequent time ticks to the empty set. Condition (3) requires that each positive integer must be contained in the interval corresponding to a tick of the temporal type. A tick i of type μ is said to be *empty* if $\mu(i) = \emptyset$. One particular consequence of the above three conditions is that the last non-empty tick (if it exists) must be mapped to an interval of the form $[i,\infty]$.

Typical calendar units, e.g., `day`, `month`, `week` and `year`, can be defined as temporal types that meet the above definition. [As an example, suppose the underlying time is measured in terms of seconds. Then the calendar unit `day` (assuming it starts on the first day of 1900) is a mapping such that `day`(1) is the set of all the seconds that comprise the first day of 1990, and `day`(2) maps to all the seconds of the second day of 1990, and so on.] An important relation regarding temporal types involves time ticks. For example, we would like to say that a particular month is within a particular year. For this purpose, we assume there is a binary (interpreted) predicate $\mathbf{IntSec}_{\mu,\nu}$ for each pair of temporal types μ and ν:

Definition For temporal types μ and ν, let $\mathbf{IntSec}_{\mu,\nu}$ be the binary predicate on positive integers such that $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **TRUE** if $\mu(i) \cap \nu(j) \neq \emptyset$, and $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **FALSE** otherwise.

In other words, $\mathbf{IntSec}_{\mu,\nu}(i,j)$ is **TRUE** iff the intersection of the corresponding absolute time intervals of tick i of μ and tick j of ν is not empty. For instance, $\mathbf{IntSec}_{\texttt{month},\texttt{year}}(i,j)$ is true iff the month i falls within the year j.

3.2 Temporal module schemes and temporal modules

We assume there is a set of attributes and a set of values called the *data domain*. Each finite set R of

[1] An interval $[i,j]$ ($[i,\infty]$, resp.) is viewed as the set of all integers k such that $i \leq k \leq j$ ($k \geq i$, resp.).

attributes is called a *relation scheme*. A relation scheme $R = \{A_1, \ldots, A_n\}$ is usually written as $\langle A_1, \ldots, A_n \rangle$. For relation scheme R, let $\mathbf{Tup}(R)$ denote the set of all mappings, called *tuples*, from R to the data domain. A tuple t of relation scheme $\langle A_1, \ldots, A_n \rangle$ is usually written as $\langle a_1, \ldots, a_n \rangle$, where $a_i = t(i)$ for each $1 \leq i \leq n$.

Definition A *temporal module scheme* is a pair (R, μ) where R is a relation scheme and μ a temporal type. A *temporal module* is a triple (R, μ, φ) where (R, μ) is a temporal module scheme and φ is a mapping, called a *time windowing* function, from $\mathcal{N}$ to $2^{\mathbf{Tup}(R)}$ such that $\varphi(i) = \emptyset$ if $\mu(i) = \emptyset$.

Intuitively, the time windowing function φ in a temporal module (R, μ, φ) gives the tuples (facts) that hold at non-empty time tick i of temporal type μ. This is a generalization of many temporal models in the literature.

A *temporal database* is a finite set of temporal modules. Throughout this paper, we assume a fixed set of temporal module schemes, which is the *database scheme*. A temporal database thus is only a different *instantiation* of the windowing functions of the fixed temporal module schemes. Furthermore, each temporal scheme is assigned a unique name. For each temporal module scheme $\mathtt{M}$, we shall use $R_{\mathtt{M}}$ and $\mu_{\mathtt{M}}$ to denote the relation scheme and temporal type, respectively. We also use $\varphi_{\mathtt{M}}$ to denote the "current" instantiation of the windowing function of scheme $\mathtt{M}$. When no confusion is possible, we shall also use $\mathtt{M}$ as the name of the current instantiation of $\mathtt{M}$. For convenience, in temporal module examples, instead of the positive integers we will sometimes use an equivalent domain. For instance, the set of expressions of the form 3/3/93:09:01:00 (**month/day/year:hour:minute:second**) will serve as such a domain.

Example 1 We view the temporal relation **ACCOUNTS** given in the introduction as a temporal module with (**ACCOUNTS**, **second**), where **ACCOUNTS** = ⟨**AcctNo**, **AccHol**, **Balance**, **AnIntRate**⟩, as its scheme. The relation in Figure 1 corresponds to the time windowing function φ defined as follows:

φ(3/3/93:09:01:00)={⟨1001, J.Smith, 1000, 3.00⟩}
φ(3/4/93:10:01:55)={⟨1001, J.Smith, 3500, 4.00⟩}
φ(3/4/93:11:00:00)={⟨1500, A.Brady, 2000, 3.00⟩}
φ(3/4/93:12:19:03)={⟨2034, T.Ford, 500, 2.50⟩}
φ(3/4/93:18:00:00)={⟨1500, A.Brady, 1500, 3.00⟩}
φ(7/3/93:09:00:00)={⟨1001, J.Smith, 4000, 4.00⟩}

Note that the above windowing function only reflects the (explicit) data stored in the relation. We will use semantic assumptions in Section 4 to give implicit data. □

Finally, we define the project operation on temporal modules that will be used in later sections. Let $\mathtt{M}$ be a temporal module and $W \subseteq R_{\mathtt{M}}$. Then $\pi^T_W(\mathtt{M})$ is the temporal module $(W, \mu_{\mathtt{M}}, \varphi')$ such that $\varphi'(i) = \pi_W(\varphi(i))$ for each $i \geq 1$. Note that π_W is the standard project operation.

3.3 The query language MQL^F

We specify the query language on temporal modules using a multi-sorted first-order logic: One sort is a generic (non-temporal) data domain, while the other sorts are temporal ones corresponding to the temporal types that we consider. Each temporal type we use gives rise to a distinct temporal sort. We shall use "temporal type" and "temporal sort" interchangeably when no confusion arises. Also, we allow certain scalar functions [EMHJ93], such as $+, -, /, *$, on the data domain. These functions are not intended as functions interpreted in the logic, but rather as external calls to the system with fixed interpretations. The formulas of this multi-sorted logic are called *MQL^F formulas* and the query language we build using the logic is called MQL^F (Multi-sorted Query Language with *F*unctions).

The range of each data variable is the data domain and the range of each temporal sort variable is $\mathcal{N}$. The *data terms* of MQL^F formulas are of the form x_0 or $f(x_1, \ldots, x_k)$, where each x_0 is a variable name or a constant of the (non-temporal) data sort, each x_i $(1 \leq i \leq k)$ is a term of the data sort, and f is a function of arity k. Thus, functions can be nested and are only for the data sort. Atomic MQL^F formulas are of the following four types: (a) $x_1 = x_2$ where x_1 and x_2 are data terms; (b) $\mathbf{IntSec}_{\mu,\nu}(t_1, t_2)$, where t_1 and t_2 are variables or constants respectively of sorts μ and ν; (c) $t_1 < t_2$ or $t_1 = t_2$, where t_1 and t_2 are variables or constants of the same temporal sort; and (d) $\mathtt{M}(x_1, \ldots, x_n, t)$ where $\mathtt{M}$ is a temporal module name of the database with n being the arity of $R_{\mathtt{M}}$, each x_i is a non-temporal variable name or constant and t a temporal variable or constant. A *MQL^F formula* is formed by boolean connectives and the existential and universal quantifiers in a usual way. As a syntactic sugar, we change the quantification of temporal sorts to the form: $\exists t{:}\mu$ and $\forall t{:}\mu$, where t is of sort μ. This syntactic sugar allows us to tell the sorts of bounded variables from the formula itself. The predicates **IntSec** are interpreted as in the previous subsection, $<$ is interpreted as the integer order, and $=$ is the standard equality.

A *MQL^F query* is of the form

$$\{x_1 \ldots x_k, t{:}\nu \mid \psi(x_1, \ldots, x_k, t)\}$$

where $x_1 \ldots x_k$ are variable names or constants of the non-temporal sort, t is temporal variable of type ν, and $\psi(x_1, \ldots, x_k, t)$ is a MQL^F formula whose only free variables are among $x_1, \ldots, x_k, t$. The formula ψ can obviously contain temporal variables of sorts different from ν, provided that they are bounded. For example,

the following is a MQL^F query on the ACCOUNTS temporal module:

$$\{x, t{:}\mathbf{month} \mid \exists y, z, w\ \exists s{:}\mathbf{second}\ (\mathtt{ACCOUNTS}(x, y, z, w, s) \wedge z \geq 2000 \wedge \mathbf{IntSec}_{\mathbf{month},\mathbf{second}}(t, s))\}$$

This query asks for the months in which an account has a balance over $2,000 for at least one second.

A MQL^F query is *correctly typed* if for all subformula $\mathtt{M}(x_1, \ldots, x_n, t)$ appearing in it, the temporal sort of t is $\mu_{\mathtt{M}}$, i.e., the temporal type of M. The above example query is correctly typed since the temporal type of the temporal module ACCOUNTS is second. With each correctly typed MQL^F query $Q = \{x_1, \ldots, x_k, t{:}\mu \mid \psi\}$, we associate an answer, denoted $Q[DB]$, in the form of a temporal module (R, μ, φ), where R is a relation scheme of arity k and φ is given as follows: $\langle a_1, \ldots, a_k \rangle$ is in $\varphi(m)$ if and only if ψ is **TRUE** when the free variables $x_1, \ldots, x_k$ and t are substituted with the constants $a_1, \ldots, a_k$ and m. Note that the truth value of the atomic formula $\mathtt{M}(y_1, \ldots, y_n, s)$ is **TRUE** iff $\langle y_1, \ldots, y_n \rangle$ is in $\varphi_{\mathtt{M}}(s)$. For example, considering the above query on a DB containing the temporal module ACCOUNTS, its answer is the temporal module $(\mathtt{AccNo}, \mathtt{month}, \varphi)$ with $\varphi(3/93) = \{1001, 1500\}$, $\varphi(7/93) = \{1001\}$, and $\varphi(i) = \emptyset$ for each i corresponding to a month different from March and July 1993.

If a MQL^F formula only has a single temporal sort, we call it a *flat-MQL^F formula*. A *flat-MQL^F query* is a MQL^F query using a flat-MQL^F formula. The following is an example of flat-MQL^F query.

$$\{x, t \mid \exists y, z, w(\mathtt{ACCOUNTS}(x, y, z, w, t) \wedge z < 1000)\}$$

4 Point-Based Assumptions

We call point-based assumptions those semantic assumptions that can be used to derive information at certain ticks of time based on the information explicitly given at different ticks of the same temporal type. This new information can be derived in different ways. One way is to assume that the values of certain attributes persist in time unless they are explicitly changed. Another way is to assume that a missing value is taken as the average of the last and next explicitly given values. Still another way is to take the sum of the last three values, etc. In this section, we give a general notion of assumptions that uses, in principle, any interpolation function to derive information from explicit values.

In the next subsection, we illustrate persistence as an example of point-based assumptions. The discussion of the persistence assumption also motivates the syntax and the semantics of general point-based assumptions.

4.1 An example: persistence assumption

A point-based assumption that seems particularly widely used in practice and in the literature is the persistence assumption. With $P_X(Y^{persis})$, we denote the assumption of the attributes XY being persistent with respect to the attributes X. This intuitively means that if we have explicit values for X and Y at a certain tick of time, these values will persist in time until we find a tick at which we have explicit values that are the same for the attributes X but different for Y. Consider our running example of the ACCOUNTS temporal module. The assumption $P_{\mathtt{AcctNo}}((\mathtt{AccHol}, \mathtt{Balance}, \mathtt{AnIntRate})^{persis})$ says that the values of these four attributes persist in time until a different value for one of the attributes AccHol, Balance, or AnIntRate with respect to the same AcctNo is found. Note that this is a reasonable assumption for the ACCOUNTS temporal module.

In the temporal database literature a notion similar to persistence is found when the value of a tuple is given for an interval $[1, uc]$ where uc is a short hand for "until changed" [WJL91, CI94]. However, that notion is not well formalized. For example, it is not clear which attributes must actually change and with respect to which other attributes. Besides having a clear semantics, persistence assumptions are more powerful, since more than one persistence assumption can be specified on the same set of attributes.

4.2 Syntax and semantics of point-based assumptions

A point-based semantic assumption relies on the use of certain methods (called *assumption methods*) to derive implicit values from explicit ones. We assume there is a fixed set of assumption methods.

Definition Let X, $Y_1, \ldots Y_n$ be pair-wise disjoint sets of attributes (i.e., $X \cap Y_i = \emptyset$ and $Y_i \cap Y_j = \emptyset$ for each i and j) and $meth_1, \ldots, meth_n$ assumption methods. Then $P_X(Y_1^{meth_1} \ldots Y_n^{meth_n})$ is called a *point-based assumption.*

The requirement that the sets X, $Y_1, \ldots, Y_n$ be pairwise disjoint in the above definition intuitively says that it is not possible to use different assumption methods for the same attribute.

If a module $M = (R, \mu, \varphi)$ has a non-empty set of associated assumptions, these assumptions actually give implied values for the involved attributes from explicit values in the windowing function φ. Formally,

Definition For each point-based assumption P with W being the set of all attributes appearing in P, there is an associated (partial) mapping f_P from temporal modules to temporal modules such that for each temporal module M with $W \subseteq R_{\mathtt{M}}$

1. $f_P(\mathtt{M})$ is a temporal module on $(W, \mu_{\mathtt{M}})$, and
2. if $f_P(\mathtt{M}) = \mathtt{M}'$, then $\pi_W(\varphi_{\mathtt{M}}(i)) \subseteq \varphi_{\mathtt{M}'}(i)$ for each positive integer i.

The first condition in the above definition specifies the scheme of the derived module. The second condition requires that the semantic assumption retains original tuples. That is, a point-based semantic assumption only adds tuples but does not change (or delete) original tuples. The above mapping f_P identifies the semantics of the assumption P. We assume that, for each assumption method *meth*, the semantics of $P_X(Y^{meth})$ where X and Y are generic set of attributes, is defined giving the corresponding mapping. As an example, the semantics of persistence could be defined as follows: For each $M = (R, \mu, \varphi)$, with $XY \subseteq R$, we have $f_{P_X(Y^{persis})}(M) = (XY, \mu, \varphi')$, where

$$\varphi'(i) = \pi_{XY}\varphi(i) \cup \pi_{XY}\{t_j \mid (\exists j < i)\,(t_j \in \varphi(j) \wedge (\forall k, t_k)\,((j < k \leq i \wedge t_k \in \varphi(k)) \Rightarrow t_k[X] \neq t_j[X]))\}$$

Note that π_{XY} is the standard projection operation.

(The semantics of an assumption that involves specific attribute names and/or multiple assumption methods can usually be derived from the semantics (i.e., the mappings) of the assumptions involving the individual methods. Due to space limitation, we omit the detailed discussion here but note the fact that the mappings associated with $P_X(Y^{meth})$ and $P_W(Z^{meth})$, for a certain method *meth*, will be the same up to renaming attributes.)

An example of a method that is different from persistence is *avg*. Applying *avg* to a numeric attribute A with respect to attributes X, it means deriving implicit tuples for XA where the value for A is taken as the average between the values of A in the last and next explicitly stored tuples that have the same values for X.

Example 2 Consider a temporal module RISKS that stores, for each account holder who received a loan from the bank, a measure of the possible risk for the bank: RISKS $= (\langle$AccHol, EstRisk$\rangle$, day, $\varphi)$ where $\varphi(2/4/93) = \{\langle$J.Smith, 2$\rangle, \langle$T.Ford, 4$\rangle\}$, $\varphi(6/4/93) = \{\langle$J.Smith, 4$\rangle, \langle$T.Ford, 3$\rangle\}$, and φ is empty for all other days. It is reasonable to introduce the assumption $P_{\texttt{AccHol}}(\texttt{EstRisk}^{avg})$ to answer queries about the Est(imated)Risk on days when the value has not been explicitly given. This assumption can be used to derive that the EstRisk value for customers J.Smith and T.Ford at all days between 2/4/93 and 6/4/93 was respectively 3 and 3.5. □

Given a temporal module M and a set of assumptions Γ, we are interested in seeing if there is a new temporal module that models all the information implied by M under Γ. Clearly, all the tuples in $f_P(M)$, for each P in Γ, should be present. Also, the new module should not include any excessive tuples. Such a new module is called a minimal closure of M under Γ. In order to formalize this notion, we first define a subsumption relation among temporal modules.

Definition Given two modules $M = (R, \mu, \varphi)$ and $M' = (R, \mu, \varphi')$ we say that M *is subsumed by* M', denoted $M \preceq M'$, if for each positive integer i, $\varphi(i) \subseteq \varphi'(i)$. The subsumption is said to be *strict*, denoted $M \prec M'$, if $M \preceq M'$ and $M \neq M'$.

Definition Let M be a temporal module and P a point-based assumption with W being all the attributes appearing in P. A temporal module $\mathtt{M}_1$ is called a *closure of* M *under* P if $\mathtt{M}_1$ is on the same temporal scheme of M, $\mathtt{M} \preceq \mathtt{M}_1$, and $f_P(\mathtt{M}) \preceq \pi^T_W(\mathtt{M}_1)$. It is called a *minimal closure of* M *under* P if $\mathtt{M}_1$ is a closure of M under P and there exist no closure $\mathtt{M}_2$ of M under P such that $\mathtt{M}_2 \prec \mathtt{M}_1$. A temporal module M′ is said to be a *closure of* M *under a set* Γ of point-based assumptions if it is a closure under each assumption P in Γ, and is said to be a *minimal closure of* M *under* Γ, if $M' \preceq M''$ for all closure M'' of M under Γ.

Notice that the minimal closure of a certain module under a set of point-based assumptions may not be unique. For example, given $\mathtt{M} = (ABC, \mu, \varphi)$ with $\varphi(1) = (a, b, c)$ and $\varphi(i) = \emptyset$ for each $i > 1$, and the assumption $P = P_A(B^{persis})$, we can find as many minimal closures of M under the assumption P, as the number of values that attribute C can take. Indeed, for each value x that C can take, the module $\mathtt{M}_x = (ABC, \mu, \varphi_x)$ with $\varphi_x(1) = (a, b, c)$ and $\varphi_x(i) = (a, b, x)$ for each $i > 1$, is a minimal closure of M under P. Here we give a sufficient condition for the existence of the unique minimal closure.

Proposition 1 *Let* Γ *be a set of point-based assumptions and* (R, μ) *a temporal scheme. If for each assumption* $P_X(Y_1^{meth_1} \ldots Y_n^{meth_n})$ *in* Γ, *we have* $XY_1 \cdots Y_n = R$, *then each temporal module on* (R, μ) *has a unique minimal closure under* Γ.

It happens that certain assumptions can be derived from a given set of assumptions. For example, we can derive $P_{\texttt{AcctNo}}(\texttt{Balance}^{persis})$ from the assumption $P_{\texttt{AcctNo}}((\texttt{AccHol}, \texttt{Balance}, \texttt{AnIntRate})^{persis})$. For reasons of space, here we do not address this issue and we refer the interested reader to [BWBJ95].

5 Query Evaluation with Point-Based Assumptions

As mentioned in the introduction, we are concerned with query answers in the presence of semantic assumptions. In this section, we assume that each temporal module M in the temporal database is associated with a set $\Gamma_{\mathtt{M}}$ of point-based semantic assumptions. We use Γ to denote the collection of all semantic assumptions. We restrict ourselves to temporal modules having a unique minimal closure under semantic assumptions. In light of Proposition 1, we assume that each semantic assumption in $\Gamma_{\mathtt{M}}$ involves all attributes in $R_{\mathtt{M}}$.

We first formally define the notion of query answer with point-based semantic assumptions.

Definition Let Q be a correctly typed MQL^F query and suppose the temporal modules in the database DB are $\mathtt{M}_1, \ldots, \mathtt{M}_m$. Then, *the answer of Q (on DB) with assumptions* Γ, denoted $Q[DB, \Gamma]$, is $Q[\overline{DB}]$, where $\overline{DB}$ is the database with the temporal modules $\overline{\mathtt{M}}_1, \ldots, \overline{\mathtt{M}}_m$ and each $\overline{\mathtt{M}}_i$ is the minimal closure of $\mathtt{M}_i$ under the assumptions $\Gamma_{\mathtt{M}_i}$.

That is, the answer of a query with assumptions is the answer of the original query on all the minimal closures of the temporal modules in the database. In other words, we first accommodate the assumptions, via minimal closures, into the temporal modules and then use the minimal closures as the temporal database to answer the query. Note that the query in question must be correctly typed since point-based semantic assumptions do not change the temporal types.

The above definition captures the purpose of semantic assumptions in answering queries. However, as mentioned in the introduction, in practice, instead of changing the temporal modules in the database to answer queries, we change the query to encompass all the semantics such that the new query on the original database will give the intended answer under assumptions. Formally,

Definition Given a query Q and semantic assumptions Γ, a query Q' is said to be a Γ-*captured version of* Q if $Q'[DB] = Q[DB, \Gamma]$ for all instantiations of the DB.

If an assumption-captured version of a query is found, in order to answer the query with assumptions, an existing system can simply evaluate this assumption-captured version in a usual way using "standard" techniques [TCG+93]. That is, the underlying query evaluation programs of an existing system need not change to accommodate semantic assumptions in its database.

In order to formalize the process that obtains a Γ-captured version, we limit the set of point-based assumptions to those that can be expressed in flat-MQL^F formulas.[2] Recall that the semantics of a point-based assumption $P = P_X(Y_1^{meth_1}, \cdots, Y_k^{meth_k})$ is a mapping f_P. Here we require that f_P is given using flat-MQL^F. In particular, we require the existence of a formula $\Psi_P^{\mathtt{M}}$ in MQL^F such that (a) only one predicate $\mathtt{M}$ appears (possibly several times) in $\Psi_P^{\mathtt{M}}$ and this predicate has arity $n+1$ where n is the number of attributes in $XY_1 \cdots Y_k$, and (b) $f_P(\mathtt{M}) = \{x_1, \ldots, x_n, t \mid \Psi_P^{\mathtt{M}}\}$ for each instantiation of the temporal module $\mathtt{M}$.

This limitation on point-based assumptions is reasonable, since flat-MQL^F is a very expressive language, and, in fact, we can specify many assumptions that are useful in practice. For example, consider the semantics of the persistence method. Let $\mathtt{M}$ be a generic temporal module such that $R_{\mathtt{M}} = XY$ with arity n. Let $\mathrm{Ind}_X^{\mathtt{M}}$ be the set of indices (positions) of attributes in X appearing in $R_{\mathtt{M}}$.[3] The flat-MQL^F formula defining persistence is:

$$\Psi^{\mathtt{M}}_{P_{X(Y^{persis})}}(w_1, \ldots, w_n, t) = \mathtt{M}(w_1, \ldots, w_n, t) \vee$$
$$(\exists t') \quad (t' < t \wedge \mathtt{M}(w_1, \ldots, w_n, t')$$
$$\wedge (\forall t'', z_1, \ldots, z_n)(t' < t'' \leq t \Rightarrow \neg(M(z_1, \ldots, z_n, t'') \wedge$$
$$(\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\ z_i = w_i))))\}$$

In the above formula, the temporal sort of all temporal variables is assumed to be $\mu_{\mathtt{M}}$. Note that for a given temporal module $\mathtt{M}$, the subformula "$(\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\ z_i = w_i$" will be written out as a conjunction of a finite number of $z_l = w_l$ for each l in $\mathrm{Ind}_X^{\mathtt{M}}$. Hence, $\Psi_P^{\mathtt{M}}$ is a flat-MQL^F formula. It is easy to check that the mapping defined through this formula corresponds to our intuitive definition of persistence and it is equivalent to the one previously defined in terms of set of tuples using the projection operation.

As explained above, once we have the definition of assumption methods in the form of the Ψ formulas identifying the corresponding mappings, we can easily obtain the formula $\Psi_P^{\mathtt{M}}$ for any assumption involving the defined methods. As an example of point-based assumption involving more than one method, consider $P_2 = P_X(Y^{avg}\, T^{persis})$, where $Y \neq \emptyset$ and $T \neq \emptyset$. Given a module $\mathtt{M}$ such that $R_{\mathtt{M}} = XYT$, this assumption intuitively says that each attribute in Y should be derived with respect to X using the *avg* (average) method while each attribute in T should simply persist (with respect to X). If $\mathrm{Ind}_X^{\mathtt{M}}$, $\mathrm{Ind}_Y^{\mathtt{M}}$, and $\mathrm{Ind}_T^{\mathtt{M}}$ are respectively the sets of indices of the attributes in $\mathtt{M}$ appearing in X,Y, and T, then the following is a the formula defining $\Psi_{P_2}^{\mathtt{M}}$. (In the formula, all the temporal variables are of the sort $\mu_{\mathtt{M}}$.)

$$\Psi^{\mathtt{M}}_{P_2}(w_1, \ldots, w_n, t) = \mathtt{M}(w_1, \ldots, w_n, t) \vee$$
$$(\exists t_1, t_2, v_1, \ldots, v_n, u_1, \ldots, u_n)\ (t_1 < t \wedge \mathtt{M}(v_1, \ldots, v_n, t_1) \wedge$$
$$(\forall t'', z_1, \ldots, z_n) \quad (t_1 < t'' \leq t \Rightarrow \neg(\mathtt{M}(z_1, \ldots, z_n, t'') \wedge$$
$$(\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\ z_i = w_i)) \wedge t < t_2 \ \wedge\ \mathtt{M}(u_1, \ldots, u_n, t_2) \wedge$$
$$(\forall t'', z_1, \ldots, z_n)(t \leq t'' < t_2 \Rightarrow \neg(\mathtt{M}(z_1, \ldots, z_n, t'') \wedge$$
$$(\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\ z_i = w_i)) \wedge (\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\ w_i = v_i = u_i \wedge$$
$$(\forall i \in \mathrm{Ind}_Y^{\mathtt{M}})\ w_i = \tfrac{v_i + u_i}{2} \wedge (\forall i \in \mathrm{Ind}_T^{\mathtt{M}})\ w_i = v_i)\}$$

We are now ready to show the process by which we obtain Γ-captured versions of given queries.

Algorithm 1 Let Q be a correctly-typed MQL^F query and Γ the collection of point-based assumptions. For each $\mathtt{M}(x_1, \ldots, x_n, t)$ appearing in Q, where each x_i is a

[2] This restriction can be relaxed if Γ-captured version can be in terms of other languages.

[3] For example, assume the attributes of $\mathtt{M}$ are $\langle A, B, C, D, E\rangle$ and $X = \{B, C, E\}$. Then $\mathrm{Ind}_X^{\mathtt{M}} = \{2, 3, 5\}$.

data variable or constant and t is a temporal variable or constant of type ν, substitute $\mathtt{M}(x_1, \ldots, x_n, t)$ with

$$\Psi^{\mathtt{M}}_{P_1}(x_1, \ldots, x_n, t) \vee \ldots \vee \Psi^{\mathtt{M}}_{P_k}(x_1, \ldots, x_n, t)$$

where $P_1, \ldots, P_k$ are all the assumptions in $\Gamma_{\mathtt{M}}$. □

Note that in the above replacement, we assume that each free variable in $\Psi^{\mathtt{M}}_{P_j}$ is substituted by the corresponding variable or constant in $\mathtt{M}(x_1, \ldots, x_n, t)$ in Q. By a simple induction on the structure of the input query, we have:

Theorem 1 *The query obtained by Algorithm 1 is a* Γ*-captured version of* Q.

Example 3 Let's consider the flat-MQL^F query given before as an example: $Q = \{x, t \mid \exists y, z, w(\mathtt{ACCOUNTS}(x, y, z, w, t) \wedge z < 1000)\}$. If $DB = \{\mathtt{ACCOUNTS}\}$ and $\Gamma = \{P_1\}$ where $P_1 = P_{\mathtt{AcctNo}}((\mathtt{AccHol}, \mathtt{Balance}, \mathtt{AnIntRate})^{persis})\}$, then Algorithm 1 gives as output the query $Q' = \{x, t \mid \exists y, z, w(\Psi^{\mathtt{ACCOUNTS}}_{P_1}(x, y, z, w, t) \wedge z < 1000)\}$. Substituting $\Psi^{\mathtt{ACCOUNTS}}_{P_1}(x, y, z, w, t)$ with the corresponding flat-MQL^F formula specifying the semantics of the persistence method, it is easy to verify that the pairs (x, t) returned as the answer $Q'[DB]$ can be viewed as the module $(\mathtt{AccNo}, \mathtt{second}, \varphi)$ with $\varphi(i) = \emptyset$ for each i before 3/4/94:12:19:03 and $\varphi(i) = \{2034\}$ for each i at and after 3/4/94:12:19:03. Note that this answer is equal to what we would obtain computing the closure of ACCOUNTS with respect to P_1 and then evaluating Q on this closure. Indeed, the minimal closure would contain all the tuples derived by the persistence, in particular those with respect to Mr.Ford's account number 2034. □

6 Interval-Based Assumptions

We call interval-based assumptions those that can be used to derive information for a certain tick of one temporal type from information at ticks of a different temporal type. The word *interval* indicates the fact that these "source" ticks must be intervals having a certain relationship (containment or intersection) with respect to the interval of the "target" tick for which we are deriving the value.

6.1 An example: liquidity

With $I(A^{\downarrow})$ we denote the assumption of the attribute A being *downward hereditary*. This intuitively means that if we have an explicit value for the attribute A at a certain tick of type μ, then for each tick of any other type that is covered by it, A has that same value. Referring to our running example, it is reasonable to assume that, if an account (AccNo) has a corresponding account holder name (AccHol) in a second, then it has the same account holder name in any millisecond, microsecond, and etc. within that second. Similarly, with $I(A^{\uparrow})$ we denote the assumption of the attribute A being *upward hereditary*. Intuitively, if we have the same value for the attribute A at contiguous ticks, that value is also the value of A for each tick of any other type that is the union of some of these ticks. In our example, if an account has the same account holder name in each second of a month, we know that the account has that account holder in that particular month. With $I(A^{\updownarrow})$ we denote the assumption of the attribute A being *liquid*, i.e., it is both downward and upward hereditary. Referring to our example, we can make the following reasonable assumption: $I((\mathtt{AccNo}, \mathtt{AccHol}, \mathtt{AnIntRate})^{\updownarrow})$.

Liquidity, as well as upward/downward heredity, can be used to answer queries that are not correctly typed. For example, the following query is not correctly typed:

$\{y, t{:}\mathtt{month} \mid \exists x, z, w(\mathtt{ACCOUNTS}(x, y, z, w, t) \wedge x = 1001\}$

This query asks for the account holder name of account 1001 in months, while the temporal type of ACCOUNTS is in seconds.

6.2 Syntax and semantics

As in point-based assumptions, an interval-based assumption relies on the use of certain "conversion" methods. For example, the liquidity assumption uses a "constant" function to derive values. We assume there is a set of conversion methods.

Definition Let $Y_1, \ldots, Y_n$ be pair-wise disjoint attribute sets, and $conv_1, \ldots, conv_n$ conversion methods. Then $I(Y_1^{conv_1} \ldots Y_n^{conv_n})$ is called an *interval-based assumption*.

In Subsection 6.1 we have illustrated three conversion methods. There are many others that can be useful. For example, given a module of type μ we assume i is a tick of a type ν and $j_1 < \cdots < j_k$ are all the ticks of type μ that are contained in tick i of ν. We may derive the value for tick i of ν using the values at tick j_1 or j_k of μ respectively. We call these conversion methods respectively *first* and *last*. Considering our running example, it is reasonable to make the assumption of converting the Balance attribute using *last*. This means that asking on the ACCOUNTS temporal module for the balance of an account for a month, it is returned the balance at the last second of that month. Note that we don't have explicit values for the last second of each month, but we have a persistence assumption to derive them. Hence, considering all the attributes in ACCOUNTS, the following is a reasonable interval-based assumption: $I((\mathtt{AccNo}, \mathtt{AccHol}, \mathtt{AnIntRate})^{\updownarrow}, \mathtt{Balance}^{last})$.

Definition For each interval-based assumption I with W being the set of all attributes appearing in I, there

is an associated (partial) mapping f_I that maps a temporal module together with a temporal type to a temporal module such that for each temporal module $\mathtt{M}$ and type ν, if $W \subseteq R_{\mathtt{M}}$, then

- $f_I(\mathtt{M}, \nu)$ is a temporal module on the scheme (W, ν), and
- $f_I(\mathtt{M}, \mu_{\mathtt{M}}) = \pi^T_W(\mathtt{M})$.

In addition, the mapping f_I must satisfy the following condition:

- Let $\mathtt{M}_1 = (R, \mu, \varphi_1)$ and $\mathtt{M}_2 = (R, \mu, \varphi_2)$ be temporal modules, and $f_I(\mathtt{M}_1, \nu) = (W, \nu, \varphi_1')$ and $f_I(\mathtt{M}_2, \nu) = (W, \nu, \varphi_2')$. Then for each $i > 0$, $\varphi_1'(i) = \varphi_2'(i)$ if $\varphi_1(j) = \varphi_2(j)$ for all j with $\mathbf{IntSec}_{\nu,\mu}(i, j) = \mathbf{TRUE}$.

The first condition says that for a given temporal module and a target temporal type, the interval-based assumption I gives a temporal module with all the attributes in I as the relation scheme of the output module and the target temporal type as the temporal type of the output module. The second condition requires that if the target temporal type coincide the temporal type of the input module, then the output module should simply be a projection of the input module. The additional condition says that the value at each tick i in the target type ν depends only on values at ticks that intersect with i.

The mapping, in the case of a liquidity assumption $I = I(W^{\downarrow})$ with regard to a module $\mathtt{M} = (R, \mu, \varphi)$ and target type ν, can be given as follows: $f_I(\mathtt{M}, \nu) = (W, \nu, \varphi')$ where for each $i > 0$

$$\varphi'(i) = \pi_W\{t \mid \forall k\,(\nu(i) \cap \mu(k) \neq \emptyset \Rightarrow t \in \varphi(k))\}$$

7 Query Evaluation with Interval-Based Assumptions

The notion of query answering with point-based assumptions in Section 5 can be extended to that with interval-based assumptions. As in Section 5, we assume there is a set $\Gamma_{\mathtt{M}}$ of interval-based assumptions for each $\mathtt{M}$ in the database, and the collection of all interval-based assumptions is denoted by Γ.

As for point-based assumptions, we restrict ourselves to assumptions involving all the attributes appearing in the modules on which they are applied. We also limit interval-based assumptions for which we can evaluate queries to those that can be specified by particular MQL^F formulas. Suppose that Γ is the set of assumptions. For each assumption I in Γ, there must be an associated formula $\Psi_I^{M,\nu}$ parametric with respect to a module $\mathtt{M} = (R, \mu, \varphi)$ and a target type ν for the conversion, such that:

$$f_I(\mathtt{M}, \nu) = \{x_1, \ldots, x_n, t{:}\nu \mid \Psi_I^{\mathtt{M},\nu}\}$$

where n is the arity of $R_{\mathtt{M}}$. The only predicate symbol involving both data and temporal sorts that can appear in $\Psi_I^{\mathtt{M},\nu}$ is $\mathtt{M}$ (possibly several times). This restriction is reasonable since MQL^F can be used to express many practically interesting interval-based assumptions. For example, liquidity is specified by:

$\Psi^{\mathtt{M},\nu}_{I(W^{\downarrow})}(x_1, \ldots, x_n, t) = (\forall t'{:}\mu)(\mathbf{IntSec}_{\mu,\nu}(t', t) \Rightarrow$
$(\exists w_1, \ldots, w_n)(\mathtt{M}(w_1, \ldots, w_n, t') \wedge (\forall j \in \mathrm{Ind}^{\mathtt{M}}_W) x_j = w_j))$

where t is of type ν, μ is the temporal type of $\mathtt{M}$, and $\mathrm{Ind}^{\mathtt{M}}_W$ is the set of indices (positions) of the attributes in W appearing in $R_{\mathtt{M}}$.

The purpose of interval-based assumptions is to derive information in terms of a different temporal type. For instance, monthly information can be derived from daily information. The advantage is that the user query does not have to be correctly typed for the system to answer. Indeed, suppose a temporal module in the database contains account interest rate information in terms of days. If the user wants to know the interest rate at a particular hour, she may query the database using hour as the temporal type for the query. The system will then use the liquidity assumption to answer the query.

Similar to the point-based assumptions, we may define query answers with interval-based assumptions and assumption-captured version of queries:

Definition Let Γ be a set of interval-based assumptions and Q a MQL^F query wrt Γ. Let $\mathcal{V}$ be the set of all the temporal types of the variables and constants appearing in Q, and for each $\mathtt{M}_i$ in DB and ν in $\mathcal{V}$, let $\overline{\mathtt{M}}_{i,\nu}$ be the temporal module $(R_{\mathtt{M}_i}, \mu_{\mathtt{M}_i}, \varphi_i)$, where for each $j \geq 1$, $\varphi_i(j) = \bigcup_{I \in \Gamma_{\mathtt{M}_i}} \varphi_{f_I(\mathtt{M}_i,\nu)}(j)$. Then *the answer of Q with assumptions* Γ (denoted by $Q[DB, \Gamma]$) is $\overline{Q}[\overline{DB}]$, where

- $\overline{DB} = \{\overline{\mathtt{M}}_{i,\nu} \mid \mathtt{M}_i \in DB \text{ and } \nu \in \mathcal{V}\}$, and
- $\overline{Q}$ is the the query obtained by changing each $\mathtt{M}_i(x_1, \ldots, x_k, t)$ to $\overline{\mathtt{M}}_{i,\nu}(x_1, \ldots, x_k, t)$ in Q, where ν is the type of t.

A query Q' is called Γ-*captured version of* Q if $Q'[DB] = Q[DB, \Gamma]$.

That is, to answer a query with assumptions Γ, one needs to (a) obtain, via the given interval-based assumptions, the temporal modules in terms of the temporal type used in the query, and (b) change the occurrences of the temporal module predicates to address the "new" temporal modules. The altered query is obviously correctly-typed, and it is then answered in a usual way over the altered database. A Γ-captured version is a query which will obtain, without changing the database, the correct answer of the original query with assumptions.

Algorithm 2 Given a MQL^F query Q on a temporal database with interval assumptions, Q is transformed in a query Q' on the original modules by substituting each occurrence of an atom $\mathtt{M}(x_1, \ldots, x_n, t)$, where each x_i is a data variable or constant and t is a variable or constant of type ν, with

$$\Psi_{I_1}^{\mathtt{M},\nu}(x_1, \ldots, x_n, t) \vee \ldots \vee \Psi_{I_k}^{\mathtt{M},\nu}(x_1, \ldots, x_n, t)$$

where $I_1, \ldots, I_k$ are all the interval assumptions in $\Gamma_{\mathtt{M}}$. □

Theorem 2 *Given a query Q and a set of interval-based assumptions Γ as inputs to Algorithm 2, the query obtained by the algorithm is a Γ-captured version of Q.*

8 Combining Point-Based and Interval-Based Assumptions

If both point-based and interval-based assumptions are present in $\Gamma_{\mathtt{M}}$, the interval-based assumptions must be applied on the minimal closure of $\mathtt{M}$ with respect to the point-based assumptions. This is quite intuitive, since values at ticks in different temporal types can be derived by conversion using values not present in the original module, but may be implied by point-based assumptions. For example, suppose the module contains the value of an attribute for the first day of the year and no value for the other days. Moreover, suppose we know that that attribute persists and that it is liquid. If we ask what is its value for the whole year, we can answer only by applying first persistence to derive that value for each day of the year, and then by applying the liquidity to derive the value for the whole year.

The answer to a query in the presence of both point-based and interval-based assumptions is defined by combining the definition of the query answering with point-based assumptions and interval-based assumptions. That is, first, the closures of the temporal modules are obtained by point-based assumptions. Then the temporal modules are changed, by interval-based assumptions, into modules in terms of the types requested by the given query. The notion of assumption-captured version can be defined easily. To obtain assumption-captured versions, one only needs to apply Algorithm 1 first and then Algorithm 2.

Theorem 3 *Let Γ be a set of point-based and/or interval-based assumptions and Q a MQL^F query. Then Algorithm 1 and 2, applied in this order, obtain a Γ-captured version of Q.*

9 Safety

As in the traditional relational calculus, we require that MQL^F queries be "domain independent" [Ull88]. Since scalar functions are used in MQL^F queries, the notion of domain independence needs to be extended. Here we adopt the notion of *embedded domain independence* of [EMHJ93]. Intuitively, a MQL^F query is embedded domain independent if its answer only depends on the *embedded domain*, where the embedded domain consists of (a) the values that appear in the database, called *active domain*, and (b) the values from a bounded number of function applications on the active domain.

Definition Given a set Γ of assumptions, a query Q is *embedded domain independent wrt* Γ if the (non-temporal) values in the answer of Q with assumptions Γ are from the embedded domain.

Embedded domain independence for MQL^F is undecidable. We thus follow [Ull88] to give a syntactic restriction on allowable MQL^F queries.[4] A MQL^F formula is said to be *safe* if it satisfies the four conditions given on page 153 of [Ull88] modified as follows: First, we ignore the temporal variables appearing in the formula. (Variables and functions of the different sorts are in fact strictly separated.) That is, the four conditions [Ull88, p.153] only apply to non-temporal variables. Second, we have to take into account the fact that we allow mathematical functions. Specifically, for condition 3 of [Ull88, p.153], we add the fact that a (data) variable x is also *limited* if it is assigned to the result of a function applied on limited variables, i.e., if $x = f(y_1, \ldots, y_m)$ is a subformula, then x is limited when $y_1, \ldots, y_m$ are all limited.

A MQL^F query $\{x_1, \ldots, x_k, t{:}\mu | \Phi\}$ is said to be *safe* if the formula Φ is safe.

It is easily seen that a safe MQL^F query is embedded domain independent.

Consider now the answer of a query with a set of semantic assumptions. A safe query does not guarantee embedded domain independence unless the assumptions satisfy certain conditions.

Definition A point-based (resp. interval-based) semantic assumption P (resp. I) is said to be *safe* if Ψ_P (resp. Ψ_I) is safe.

Persistence and heredity assumptions are easily seen as safe point-based and interval-based assumptions, respectively. Since a semantic assumption can be represented as a MQL^F query, its safety implies embedded domain independence.

Proposition 2 *If Q is a safe query and Γ a set of safe assumptions, then Q is embedded domain independent with respect to Γ.*

Proposition 3 *Let Q be a query and Γ a set of assumptions. Suppose each assumption in Γ, as well*

[4] A more elaborate criterion called "embedded allowed" appeared in [EMHJ93].

as Q, is safe. Then Q', obtained by the transformation from Q by Algorithms 1 and 2, is embedded domain independent.

From the above two propositions we conclude that to ensure the embedded domain independence of the original query with respect to assumptions, it is sufficient to check the safety of the original query if all the assumptions are assumed to be safe.

Unlike the traditional relational databases, however, the finiteness of (non-temporal) values appearing in an answer does not guarantee the "finiteness" of the answer. Many natural semantic assumptions lead to infinite answers. In Example 3, the answer to the query $Q = \{x, t \mid \exists y, z, w\,(\texttt{ACCOUNTS}(x, y, z, w, t) \wedge z < 1000)\}$, asking for the accounts (and the times) that have a balance under \$1,000, is infinite. Indeed, the windowing function φ of the resulting module is non-empty ($\varphi(i) = \{2034\}$) for an infinite number of ticks. Note that the number of different (non-temporal) values in the answer is finite.

The above infinite answer, however, shows a particular characteristics, namely, for each time i which is after a certain fixed time T, the value (set of tuples) associated with i ($\varphi(i)$) is always the same. We call such an infinite temporal module "eventually uniform."

Definition We say that a module is *eventually uniform* if after a certain tick its windowing function always gives the same value. Formally, if $\mathtt{M} = (R, \mu, \varphi)$, $\mathtt{M}$ is eventually uniform if there exist t_{max} such that $\varphi(i) = \varphi(t_{max})$ for each $i > t_{max}$.

Note that a representation equivalent to an eventually uniform module is used in almost all temporal extensions of relational databases. A typical way of representing the eventual uniformity is associating to a value/tuple the pseudo-interval $[c_1, \infty]$, where c_1 is a time constant.

In this section, we show that every flat-MQL^F query gives eventually uniform answers if the temporal modules in the database are all eventually uniform and each assumption is expressed using a flat-MQL^F formula. We start, however, with a more general notion and result for MQL^F queries.

Definition We say that a temporal module is *1st-order finitely partitioned* (*1fp-module* for short) if there always exists a partition of its ticks into a finite number of sets such that its windowing function always gives the same finite set of tuples within these sets. Moreover each such set of ticks can be specified by a first-order formula with only temporal variables and predicates **IntSec** and $<$.

Theorem 4 *The answer to a MQL^F query on a database with assumptions is always 1st-order finitely partitioned if each module in the database is 1st-order finitely partitioned.*

Theorem 5 *Let Γ be a set of assumptions expressed in terms of flat-MQL^F formulas. The answer to a query in flat-MQL^F on a database with assumptions Γ is always eventually uniform if each temporal module in the database is eventually uniform.*

Moreover, the value t_{max} of the tick after which the answer is always the same can be computed for each query Q and input modules $\mathtt{M}_1, \ldots, \mathtt{M}_k$ with assumptions Γ.

10 Conclusion

This paper introduced the notion of semantic assumptions for temporal databases. Semantic assumptions allow a compact representation of potentially infinite temporal data; they can be used to compute values not explicitly given and to convert data from one temporal type to another. While some of these assumptions have been extensively used in the literature on temporal databases, they were not adequately formalized. This paper gives such a formalization considering two very general classes of assumptions: a) point-based assumptions used on temporal databases with a single temporal type, and b) interval-based assumptions that are specific for databases allowing different temporal types.

We impose that each assumption involves all the attributes of the scheme to which it is applied to ensure that the minimal closure of the temporal database is unique. This condition can be seen as a limitation, but it can be relaxed for the purpose of answering queries. One option is to introduce "don't care" values into the query language. We omit the details due to the space limitation.

Another interesting issue concerns the expressiveness of the query language. Even though MQL^F is a powerful language, there are certain assumptions that cannot be specified. An example of derivation method that cannot be specified is *weighted sum*, where a value is computed as the weighted sum of a set of values using as weights the tick numbers that attach to the values of the set. Indeed, MQL^F does not allow terms of temporal sorts as arguments of functions. There are also interval-based assumptions that cannot be expressed in MQL^F. For example, consider a conversion function saying that values in terms of *months* can be derived from values in terms of *years* by dividing it by 12. The effect of an assumption using this conversion should be limited to *months* and *years*, but there is no way to do so using MQL^F. To take into account this kind of conversion function it is necessary to introduce the notion of an interval-based assumption restricted to a subset of the temporal types of the database. Extending the expressiveness of our query language along this direction should not affect the results reported in the paper.

In addition to the formalization of semantic as-

sumptions, a major contribution of the paper is a simple method to transform a query on a temporal database with assumptions into a new query on the same database but without assumptions, so that it is not necessary to compute the minimal closure of each module in the database to obtain the answer. The same algorithms can probably be used to check constraint satisfaction on temporal databases with assumptions. For example, dynamic integrity constraints are specified in [Cho92] using Temporal Logic (TL); it is easy to show that every formula in TL can be translated into an equivalent formula in MQL^F.

Beyond the topics we mention above, other issues that are worth investigating include: (1) How does a system efficiently evaluate a query with assumptions? Since an assumption (such as persistence) may be used on many relations and, by its nature, all queries addressed to such relations must be evaluated with the assumption, it is desirable (and also possible) for the system to exploit the properties of the assumptions in order to provide more efficient evaluations. In order to accomplish this, we need to (2) further categorize semantic assumptions for a better understanding. Lastly, (3) How do we incorporate semantic assumptions into the evaluation of a practical query language (such as SQL)?

Acknowledgement

The authors wish to thank the anonymous referees for their thorough reviews and constructive suggestions.

References

[AKPT91] J. F. Allen, H. Kautz, R. Pelavin, and J. Tenenberg. *Reasoning about Plans.* Morgan-Kaufman, 1991.

[BWBJ95] C. Bettini, X. Wang, E. Bertino, and S. Jajodia. Introducing assumptions in temporal databases. Manuscript, 1995. GMU.

[CCT94] J. Clifford, A. Croker, and A. Tuzhilin. On completeness of historical relational query languages. *ACM TODS*, 19(1):64–116, March 1994.

[Cho92] J. Chomicki. History-less checking of dynamic integrity constraints. In *Proc. Data Engineering*, pages 557–564, February 1992.

[CI94] J. Clifford and T. Isakowitz. On the semantics of (bi)temporal variable databases. In *Proc. EDBT*, pages 215–230, March 1994.

[CT85] J. Clifford and A.U. Tansel. On an algebra for historical relational databases: Two views. In *Proc. ACM SIGMOD*, pages 247–265, May 1985.

[CW83] J. Clifford and D.S. Warren. Formal semantics for time in databases. *ACM TODS*, 8(2):214–254, June 1983.

[DM87] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence J.*, 32:1–55, 1987.

[EMHJ93] M. Escobar-Molano, R. Hull, and D. Jacobs. Safety and translation of calculus queries with scalar functions. In *Proc. ACM PODS*, pages 253–264, May 1993.

[Sho87] Yoav Shoham. Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence J.*, 33:89–104, 1987.

[Sno84] R. Snodgrass. The temporal query language TQuel. In *Proc. ACM PODS*, pages 204–212, April 1984.

[SS87] A. Segev and A. Shoshani. Logical modeling of temporal data. In *Proc. ACM SIGMOD*, pages 454–466, May 1987.

[Tan87] A. U. Tansel. A statistical interface for historical relational databases. In *Proc. Data Engineering*, pages 538–546, February 1987.

[TCG+93] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors. *Temporal databases: Theory, design, and implementation.* Benjamin/Cummings, 1993.

[Ull88] J. D. Ullman. *Priciples of Database and Knowledge-base Systems.* Computer Science Press, 1988.

[WBBJ94] X. Wang, C. Bettini, A. Brodsky, and S. Jajodia. Logical design for temporal databases with multiple temporal types. Technical Report ISSE-TR-94-111, GMU, 1994.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Springer-Verlag LNCS, Vol. 498*, pages 124–140, 1991.

[WJS95] X. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal modules: An approach toward federated temporal databases. *Information Sciences*, 82:103–128, 1995. A preliminary version of this paper appeared in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, DC, May 1993, pp. 227–236.