Temporal Semantic Assumptions and Their Use in Database Query Evaluation*

Claudio Bettini X. Sean Wang Sushil Jajodia

Technical Report: ISSE-TR-95-105
Department of Information and Software Systems Engineering
George Mason University

June 13, 1995

Abstract

Temporal data explicitly stored in a temporal database are often associated with certain semantic assumptions. Each assumption can be viewed as a way of deriving implicit information from the explicitly stored data. Rather than leaving the task of deriving (possibly infinite) implicit data to application programs, as is the case currently, it is desirable that this be handled by the database management systems. To achieve this, 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). In order to incorporate semantic assumptions into query evaluation, this paper introduces a translation procedure that converts a user query 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 also investigates the finiteness (safety) of user queries and system queries.

1 Introduction

A prominent feature of temporal information is its richness in semantics associated with its temporal domain. When querying a temporal database, a user naturally assumes some "usual" semantics on stored temporal data. For instance, she expects that her bank account balance persists, i.e., the balance stays the same unless a transaction—deposit, withdrawal or accrual of interest—is performed. Therefore, if she wishes to find the balance at a particular time and no balance amount is stored for that time, she looks for the balance of the last transaction that was performed before the time in question. Semantic assumptions may also involve different temporal types (time granularities). For example, when the user

*A preliminary version of this paper appeared in the 1995 Proceedings of the ACM SIGMOD International Conference on Management of data. This work was 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 in part while visiting George Mason University. Work of Wang was also supported by the NSF grant IRI-9409769. Bettini is with the Dipartimento di Scienze dell'Informazione, University of Milano, 20135 Milano, Italy, and Wang and Jajodia are with the Department of Information and Software Systems Engineering, George Mason University, Fairfax, VA 22030–4444. Emails: bettini@dsi.unimi.it and {xywang, jajodia}@isse.gmu.edu.

asks for the account holder of a certain account in a particular month and account holders are stored in terms of days, she naturally assumes that the answer is someone who is the account holder of that account on all the days *within* that month.

Researchers have long realized such richness of semantics in temporal data [CT85, SS87, Tan87] and provided various operators in different query languages for the users to code the semantic assumptions into queries [SS87, Tan87]. For the aforementioned balance query, the user would use a "last" (or similar) operator in her query to retrieve the appropriate balance.

We argue, however, that temporal semantics should be an integral part of temporal databases and users should not be burdened with having to incorporate these semantics in their queries. In the account balance example, the user should be able to query the database directly about the balance at any particular time, instead of having to code the semantics of the balance into the query. The system should be able to answer the query appropriately according to the semantics. To provide such an ability, the system has to

- include a precise formalization of the temporal semantics, and
- evaluate queries according to these semantics.

In this paper, we provide a framework to incorporate temporal semantics into temporal databases and investigate a method to evaluate queries with semantics.

Consider the bank account example in more detail. Assume the temporal relation `ACCOUNTS` records account numbers (`AcctNo`), account holder (`AccHol`), account balance (`Balance`), annual interest rate (`AnIntRate`), and the associated time (`Time`), i.e., the time when the reported values become true. We can assume that new tuples are added to this relation when an event occurs such as the opening of an account, a deposit, a withdrawal, or a change in interest rates. The values of `Time` are timestamps consisting of the date (month/day/year) concatenated with the time of the day (up to seconds). (Here and in the rest of the paper timestamps represent *valid time*; extensions to include *transaction* time and other temporal dimensions are not considered.)

An instance of the `ACCOUNTS` relation is shown in Figure 1.

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`

Since we store new tuples only when some of the attribute values change, the relation does not contain an explicit tuple for every second of each day for an account. However, if asked about the balance of Mr. Smith's account at noon of March 4, 1993, we could answer without hesitation that the balance was $3,500. This is because we assume that nothing has changed for that account since the last transaction before that time. Here we say that the attribute `Balance` satisfies a semantic assumption called *persistence* for each account. Another example of persistence is given by the attribute `AccHol` that is intuitively persistent for each account.

Other assumptions apply when different temporal types (or granularities) are considered. For example, the attribute `AnIntRate` can be classified as *liquid*, meaning that it satisfies the following properties:

It is downward-hereditary: If the value of `AnIntRate` is 3.00 for a month, we can assume that its value is 3.00 for each day of that month.

It is upward-hereditary: If the value of `AnIntRate` is 3.00 for each day of a month, we can assume that its value is 3.00 for that month.

The terms "liquid" and "downward/upward hereditary" are borrowed from the temporal reasoning community [Sho87].

Note that not all attributes are persistent and/or liquid. For example, consider a relation with the attributes ⟨`AcctNo, Amount, second`⟩ that stores the time and the amount of each deposit or withdrawal to a certain account. The `Amount` attribute is obviously neither persistent nor liquid.

The aforementioned semantic assumptions, namely persistence and liquidity, are among the most common ones that have been identified by the temporal reasoning community. There are other semantic assumptions that arise quite naturally in databases. For example, if a database stores annual salaries of the employees, the monthly salary can easily be determined by dividing the annual salary by 12.

Contribution of the paper

Recognizing and formalizing that a group of attributes satisfies some semantic assumption is very useful in answering queries about values of attributes at times (of same or different temporal types) for which a value is not explicitly stored. One of the contributions of this paper is to formalize the notion of semantic assumptions for the purpose of query evaluation.

In this paper, each semantic assumption is viewed as a function for deriving implicit information from explicit information by applying one or more information generation procedures. Such generation procedures are formally captured by our notion of assumption methods. Intuitively, an assumption method derives implicit information using some specific rule. For example, "persistence" can be regarded as one such rule that uses the latest stored value as the current value for an attribute. In Figure 1, by applying the persistence method on the `Balance` attribute, we obtain \$1,000 to be the balance at the time 3/3/93:09:01:01, 3/3/93:09:01:02, 3/3/93:09:01:03, 3/3/93:09:01:04, and so on. The function for a semantic assumption that uses possibly different methods to generate information for different attributes, is then a "composition" of the rules of the appropriate assumption methods.

We denote by DB the database and by $\overline{DB}$ the database that contains, in addition to DB, the data generated by applying all semantic assumptions. A user query on a database DB is viewed as a query on $\overline{DB}$. If the query asks for the balance of account 1001 at noon of March 4, 1993, it retrieves values from $\overline{DB}$, and gives \$3,500 as the answer.

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` for each calendar unit. Not only an enormous effort is required to manage all these derived values, but this effort may be wasted since users may not be interested in knowing the interest rate for all possible calendar units. A similar problem exists for the `Balance` attribute.

Our solution is to translate the user query into a query that incorporates the semantic assumptions. As shown in Figure 2, a user query Q is viewed as a query on $\overline{DB}$. Instead of using the 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 $\overline{DB}$ ($Q[\overline{DB}]$). In formula, we require that

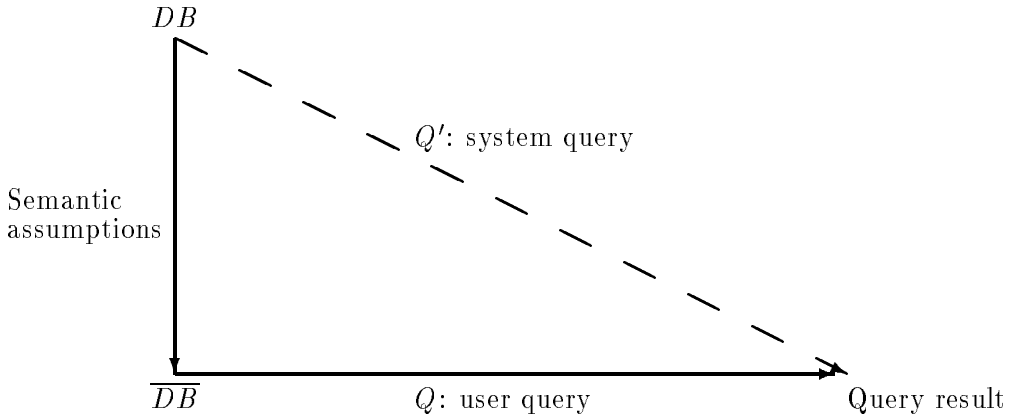

Figure 2: Does there exist a Q' such that $Q'[DB] = Q[\overline{DB}]$?

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

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 [WJS93] 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 aforementioned system query Q' always exists if all semantic assumptions are specified by MQL^F and all user queries are in MQL^F. We provide automatic derivation of Q'.

Another contribution of the paper is in its analysis of safety issues of the temporal database query language MQL^F. Persistence, and other semantic assumptions, naturally gives rise to infinite information. For example, the persistence of the `Balance` attribute dictates that the relation in Figure 1 generates infinitely many tuples: For account 1001, there will be one tuple for each second after 7/3/93:09:00:00. There are an infinite number of seconds. 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 is not analyzed formally in a general setting. In this paper, we study the safety requirement of query languages that yields only eventually uniform results and other general safety criteria.

In summary, the contribution of the paper is three-fold: (1) Semantic assumptions are formalized for the purpose of database query evaluation; (2) A paradigm for query evaluation is introduced to incorporate semantic assumptions; and (3) Safety issues are analyzed in this more general setting.

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 were perhaps recognized first by Clifford and Warren [CW83]. The earliest systematic study was 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 were 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] pointed out the use of interpolation in temporal databases; however, the query evaluation was not formalized. Clifford and Isakowitz [CI94] dealt with formalizing the semantics of *variables* that many temporal data models employ to denote various intuitive semantic assumptions. The work clarified many vague, although intuitive, notions. However, [CI94] did 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.

Organization

The rest of the paper is divided into 8 sections. Section 2 defines our temporal data model. In Section 3, we introduce the notion of a general class of semantic assumption: point-based semantic assumptions. Persistence is a special case of the point-based assumptions. Several important properties of point-based assumptions, and the persistence assumption in particular, are investigated. Query evaluation with respect to point-based assumptions is discussed in Section 4. Interval-based assumptions that include the liquidity assumption are formalized in Section 5 and query evaluation with respect to such assumptions is studied in Section 6. In Section 7, the above two kinds of assumptions are combined in evaluating queries. Safety issues are discussed in Section 8. Finally, the paper is concluded with some remarks on the results of the paper and on future research directions in Section 9.

2 Data Model

The data model used in this paper is based on that introduced in [WJS93, 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.

2.1 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 such as `day`, `month`, `week` and `year` can be defined as temporal types. 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.

There is a natural "finer-than" relation among temporal types. The temporal type μ is said to be finer than the temporal type ν if for each tick of μ, the corresponding time interval is "entirely covered" by the time interval of some tick of ν. Thus, for example, `day` is finer than `week` and `month` is finer than `year`. Formally, we have:

Definition Let μ_1 and μ_2 be temporal types. Then μ_1 is said to be *finer than* μ_2, denoted $\mu_1 \preceq \mu_2$, if for each i, there exists j such that $\mu_1(i) \subseteq \mu_2(j)$.

It is easily seen that $\preceq$ is a partial order. However, it is not a total order since, for example, `week` and `month` are incomparable (i.e., `week` is not finer than `month`, and `month` is not finer than `week`). In fact, the set of all temporal types is a complete lattice with respect to the finer-than relation [WJS93].

Another 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.

[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.).

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. Thus, for example, $\mathbf{IntSec}_{\mathtt{month},\mathtt{year}}(i,j)$ is true iff the month i falls within the year j.

It is important to emphasize that a realistic system can treat infinite temporal types[2] and their corresponding **IntSec** relations only if these infinite types have finite representations. Various periodical descriptions, e.g., [KSW90, NS92], are possible but outside the scope of this paper.

2.2 Temporal module schemes and temporal modules

The temporal modules were presented in [WJS93, WBBJ94] as a unified interface for accessing different underlying temporal information systems. They can be viewed as "abstract temporal databases" [Cho94] or a "conceptual data model" [JSS93]. It is believed that the concepts and the results of this paper are readily translated in terms of other temporal data models.

We assume that there is a set of attributes and a set of values called the *data domain*. Each finite set R of 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 τ of relation scheme $\langle A_1, \ldots, A_n \rangle$ is usually written as $\langle a_1, \ldots, a_n \rangle$, where $a_i = \tau(i)$ 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 $\mu(i) = \emptyset$ for each i, and $\bigcup_{i \geq 1} \varphi(i)$ is a finite set.

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 `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 `M`. We use M as the name of the current instantiation of `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⟩}

[2] A temporal type is said to be *infinite* if it has an infinite number of non-empty ticks.

Note that the above windowing function only reflects the (explicit) data stored in the relation. We will use semantic assumptions in Section 3 to give implicit data. □

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

2.3 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 Functions).

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 scheme 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. An *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.

An *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 a temporal variable of type ν, and $\psi(x_1, \ldots, x_k, t)$ is an 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 they are bounded. The following is an example MQL^F query on the `ACCOUNTS` temporal module:

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

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

An 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 $\mu_{\mathtt{M}}$, namely the temporal type of $\mathtt{M}$. 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, ..., x_k and t are substituted with the constants a_1, ..., 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 $\varphi_{\mathtt{M}}(s)$. If we consider the aforementioned example query on a DB containing the temporal module ACCOUNTS of Example 1, its answer is the temporal module $(\mathtt{AcctNo}, \mathtt{month}, \varphi)$ 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 of 1993.

If a query is not correctly typed, there is no answer associated with the query. In Section 6, we will show how to answer incorrectly typed queries by using interval-based semantic assumptions.

If an MQL^F formula only has a single temporal sort, we call it a *flat-MQL^F formula*. Alternatively, we view flat-MQL^F as a 2-sorted fragment of MQL^F, where all temporal variables and temporal constants are of the same temporal type. Since there is a single type μ, the only predicate of the form **IntSec** is $\mathbf{IntSec}_{\mu,\mu}$, and $\mathbf{IntSec}_{\mu,\mu}$ is equivalent to $=$. Hence, in a flat-MQL^F formula, we assume that **IntSec** predicates do not appear. A *flat-MQL^F query* is an MQL^F query in which the formula is a flat-MQL^F formula. If a flat-MQL^F query is correctly typed, all temporal constants and variables appearing in the query have the same sort and we may omit the syntactic sugar on the quantifiers on temporal variables and the type declaration of the free temporal variable when their type is clear from the context. The following is an example of flat-MQL^F query.

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

Assuming this query is correctly typed, then the type of t is $\mu_{\mathtt{ACCOUNTS}}$ which is second. This query asks for the time when an account has a balance lower than \$1,000.

3 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. Such derivation can be done in a variety of 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 point-based assumptions that uses, in principle, any interpolation function to derive information from explicit values.

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

3.1 An example: Persistence assumption

A point-based assumption that is 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. More formally, let $M = (R, \mu, \varphi)$ be a temporal module and $P_X(Y^{persis})$ be a persistence assumption. Then, we say that a tuple t on XY is *derived* by this assumption for tick i if there exists $j < i$ such that (1) there exists $t'' \in \varphi(j)$ such that $t = t''[XY]$, and (2) for all k, $j < k \leq i$, and tuple $t' \in \varphi(k)$, we have $t'[X] \neq t[X]$. To be complete, we also keep all the

information (projected on XY) in the original temporal module. Hence, a persistence assumption on a temporal module gives all the original information (projected on XY) plus all the derived information.

Consider our running example of the ACCOUNTS temporal module. The assumption

$$P_{\texttt{AcctNo}}((\texttt{AccHol}, \texttt{Balance}, \texttt{AnIntRate})^{persis})$$

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, the notion of until changed is not well formalized. For example, it is not clear which attributes must actually change to be qualified as "changed". 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.

In the next subsection, we formally define the syntax and the semantics of point-based assumptions, of which the persistence assumption is an example.

3.2 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. A method is used to derive a value of an attribute by "interpolating" values of the same attribute at different ticks of time. The expression $P_X(Y^{meth})$ denotes the assumption using method $meth$ to derive implicit values of attributes XY with respect to attributes X. We assume there is a fixed set of assumption methods.

We now give the syntax of semantic assumptions.

Definition Let X, Y_1, ..., 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 $i \neq j$) and $meth_1$, ..., $meth_n$ assumption methods. Then $P_X(Y_1^{meth_1} \ldots Y_n^{meth_n})$ is called a *point-based assumption*. The attribute set X is called the *base* of the assumption.

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

Before defining the semantics of our semantic assumptions, we need to introduce the semantics of assumption methods.

Definition (Semantics of assumption methods)
For each assumption method $meth$, there is an associated mapping $meth()$ from two sets of attributes and a temporal module to a temporal module such that, for all attribute sets X and Y and temporal module $M = (R, \mu, \varphi)$, with $XY \subseteq R$, the following conditions are satisfied:

1. $meth(X, Y, M)$ is a temporal module on (XY, μ);
2. if $meth(X, Y, M) = (XY, \mu, \varphi')$, then $\pi_{XY}(\varphi(i)) \subseteq \varphi'(i)$ for each positive integer i;
3. for each set Z of attributes with $Z \subseteq Y$, $meth(X, Z, \pi^T_{XZ}(M)) = \pi^T_{XZ}(meth(X, Y, M))$; and
4. for each value x on X, $meth(X, Y, \sigma^T_{X=x}(M)) = \sigma^T_{X=x}(meth(X, Y, M))$.

The first condition in the above definition specifies the scheme of the derived modules. The second condition requires that each derived module contains all the original tuples. That is, a semantic method can only be used to add tuples but not to change (or delete) original tuples. The third condition in the definition requires that each method derives values for a certain attribute independently from values of other attributes. The final condition requires that each method derives values for a certain attribute by considering only tuples in the original module having the same values for X.

Although the semantics of an assumption method is given as a mapping, it is usually described by a syntactic expression in practice. Any formal language that is sufficiently expressive can be used for this purpose. In Section 4 we give a formalism based on MQL^F to specify the semantics of assumption methods.

An example of a method that is different from persistence is *avg*. By applying *avg* to a set of numeric attributes Y with base attributes X, we derive the implicit tuples on XY where the value for each $A \in Y$ 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: $\texttt{RISKS} = (\langle \texttt{AccHol}, \texttt{EstRisk} \rangle, \texttt{day}, \varphi)$ where $\varphi(2/4/93) = \{\langle \text{J.Smith}, 2\rangle, \langle \text{T.Ford}, 4\rangle\}$, $\varphi(6/4/93) = \{\langle \text{J.Smith}, 4\rangle, \langle \text{T.Ford}, 3\rangle\}$, and φ is empty for all other days. Then $avg(\{\texttt{AccHol}\}, \{\texttt{EstRisk}\}, \texttt{RISKS})$ is the temporal module that contains the original information as well as the following: For each day between 2/4/93 and 6/4/93, the `EstRisk` values for customers J.Smith and T.Ford were respectively 3 and 3.5. □

Having defined the semantics of assumption methods, we can now give the semantics of semantic assumptions.

Definition (Semantics of point-based assumptions)
For each point-based assumption $P = P_X(Y_1^{meth_1} \ldots Y_k^{meth_k})$, there is an associated (partial) mapping f_P from temporal modules to temporal modules such that for each temporal module $M = (R, \mu, \varphi)$ with $XY_1 \ldots Y_k \subseteq R$, if $meth_j(X, Y_j, M) = (XY_j, \mu, \varphi_j)$ for each $j = 1, \ldots, k$, then $f_P(M) = (XY_1 \ldots Y_k, \mu, \varphi')$ where $\varphi'(i) = \varphi_1(i) \bowtie \ldots \bowtie \varphi_k(i)$ for each $i \geq 1$.

Thus, the semantic assumption $P_X(Y_1^{meth_1} \ldots Y_k^{meth_k})$ combines (via natural join) the implicit information on attributes XY_1, ..., XY_k supplied by the methods $meth_1$, ..., $meth_k$, respectively.

A consequence of the above definition is the fact that in general, the semantics associated with $P_X(Y_1^{meth} Y_2^{meth})$ is *not* always the same as that of $P_X((Y_1Y_2)^{meth})$. Indeed, consider the temporal module $M = (\langle A, B, C\rangle, \mu, \varphi)$, where $\varphi(1) = \{\langle 1,2,3\rangle, \langle 1,3,4\rangle\}$, and the two semantic assumptions $P_1 = P_A(B^{persis} C^{persis})$, $P_2 = P_A((BC)^{persis})$. Let $f_{P_1}(M) = (\langle A, B, C\rangle, \mu, \varphi_1)$ and $f_{P_2}(M) = (\langle A, B, C\rangle, \mu, \varphi_2)$. It is easily seen that $\langle 1,2,4\rangle$ is in $\varphi_1(2)$, but not in $\varphi_2(2)$.

We consider this as natural since it is likely that the reason that the two sets (Y_1 and Y_2) of attributes are written separately is because their values are derived independently. However, the base X of a point-based assumption is often a temporal key (i.e., there do not exist two distinct tuples that belong to the same tick yet have the same X values), in which case the semantics of $P_X(Y_1^{meth} Y_2^{meth})$ and $P_X((Y_1Y_2)^{meth})$ are the same. Indeed, by condition 3 of assumption methods semantics, the temporal modules obtained by $meth(X, Y_1, M)$ and $meth(X, Y_2, M)$ are the projections of $M' = meth(X, Y_1Y_2, M)$. Since X is a temporal key, the join of the two projections, (i.e., the semantics of $P_X(Y_1^{meth} Y_2^{meth})$) is the same module M' (i.e., the semantics of $P_X((Y_1Y_2)^{meth})$).

3.3 Properties of temporal modules with assumptions

Given a temporal module M and a set of assumptions , , we would like to know if there is a new temporal module that models all the information implied by M under , . Clearly, all the tuples in $f_P(M)$, for each P in , , should be present; however, the new module should not include any extraneous tuples. We call such a module minimal closure of M under , . 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 as the set of all the attributes that appear in P. A temporal module M_1 is called a *closure of* M *under* P if M_1 is on the same temporal scheme of M, $M \preceq M_1$, and $f_P(M) \preceq \pi^T_W(M_1)$. M_1 is called a *minimal closure of* M *under* P if it is a closure of M under P and there does not exist closure M_2 of M under P such that $M_2 \prec M_1$. A temporal module M' is said to be a *closure of* M *under a set* , of point-based assumptions if it is a closure under each assumption P in , , and is said to be a *minimal closure of* M *under* , , if there does not exist closure M'' of M under , such that $M'' \prec M'$.

Notice that the minimal closure of a certain module under a set of point-based assumptions may not be unique. For example, given $M = (ABC, \mu, \varphi)$ 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 $M_x = (ABC, \mu, \varphi_x)$ 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 , be a set of point-based assumptions and (R, μ) a temporal scheme. If for each assumption $P_X(Y_1^{meth_1} \ldots Y_k^{meth_k})$ in , , we have $XY_1 \cdots Y_k = R$, then each temporal module on (R, μ) has a unique minimal closure under , . Furthermore, if the minimal closure of a temporal module M under , is (R, μ, φ), and the minimal closure of M under each assumption P in , is (R, μ, φ_P), then $\varphi(i) = \bigcup_{P \in \Gamma} \varphi_P(i)$.

The opposite is not necessarily true even if we have only persistence assumptions. Consider for example the temporal module $M = (ABC, \mu, \varphi)$ with $\varphi(1) = (1, 1, 1)$ and $\varphi(i) = \emptyset$ for all other ticks. Suppose we have two assumptions, $P_{AB}(C^{persis})$ and $P_C(B^{persis})$. Since $P_C(B^{persis})$ does not include all the attributes in M, it violates the condition of the above proposition. However, M does have the unique minimal closure $M' = (ABC, \mu, \varphi')$, where $\varphi'(i) = (1, 1, 1)$ for all i.

We use minimal closures to develop two useful notions. The first notion deals with the question whether two temporal modules are equivalent under assumptions. If there are no semantic assumptions, two temporal modules are equivalent if and only if they have the same schemes, same temporal types and same windowing functions, i.e., they are identical temporal modules. However, when semantic assumptions are present, two temporal modules may not be identical, yet they may contain the same information. To see this, consider the temporal scheme $(\langle$`AccHol,Address`$\rangle$, `month`$)$ and the two modules M_1 and M_2 on this scheme. For the first month, $\varphi_{M_1}(1) = \varphi_{M_2}(1) =$ {(`J.Smith, 16 Fifth Av.,NY`)}, while for the second month $\varphi_1(2) =$ {(`A.Brady, 5 Old St.,NY`)} and $\varphi_{M_2}(2) = \varphi_{M_1}(1) \cup \varphi_{M_1}(2)$. These modules are not identical, since $\varphi_{M_1}(2) \neq \varphi_{M_2}(2)$. However, they are equivalent under the assumption $P_{\texttt{AccHol}}(\texttt{Address}^{persis})$, since they have identical minimal closures under that assumption.

Definition Given two modules $M = (R, \mu, \varphi)$ and $M' = (R, \mu, \varphi')$, we say that M is *equivalent* to M' under assumptions , if they have identical sets of minimal closures.

Since there may be different modules that are equivalent under a set of assumptions, it is interesting to see if there is a minimum temporal module (with respect to the subsumption relation) among these equivalent modules. In practice, we may choose to use such a minimum module to save storage space.

Definition (Minimal Representation)
Let , be a set of point-based assumptions and M a temporal module. A temporal module M' is said to be *a minimal representation of* M *under* , if M' is equivalent to M under , , and there is no temporal module M'' such that M'' is equivalent to M and $M'' \prec M'$.

The second notion we develop deals with implicit assumptions. It is possible to derive implicit assumptions from a given set of explicit ones. For example, given the assumption $P_{\texttt{AcctNo}}((\texttt{AccHol}, \texttt{Balance}, \texttt{AnIntRate})^{persis})$ we can derive $P_{\texttt{AcctNo}}(\texttt{Balance}^{persis})$. Intuitively, given an assumption P and a set , of assumptions, we will say that P is implied by , if, for any module, any implicit information that can be derived from P can also be derived from , . Formally,

Definition (Implied Assumption)
A point-based assumption P is implied by a set of assumptions , (we write , $\models P$) if given an arbitrary module M, each closure of M under , is also a closure of M under P.

Proposition 2 The following derivation rule is sound for all $meth_1$, ..., $meth_k$:

- $P_X(Y_1^{meth_1}, \ldots, Y_k^{meth_k})$ implies $P_X(W_1^{meth_1}, \ldots, W_k^{meth_k})$ if $W_j \subseteq Y_j$ for each j with $1 \leq j \leq k$.

An interesting property of persistence assumption is that there exists a sound and complete derivation rule for persistence assumptions.

Theorem 1 The following derivation rule is both sound and complete for persistence assumptions:

- If $P_X(Y^{persis})$, then $P_Z(W^{persis})$ for all $Z \subseteq X$ and $W \subseteq (X \setminus Z) \cup Y$.

4 Query Evaluation with Point-Based Assumptions

In this section, we assume that each temporal module M in the temporal database is associated with a set , $_M$ of point-based semantic assumptions. We use , 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 , $_M$ involves all the attributes in $R_{\texttt{M}}$.

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 $M_1, \ldots, M_m$. Then, *the answer of* Q *(on DB) with assumptions* , , denoted $Q[DB, ,]$, is $Q[\overline{DB}]$, where $\overline{DB}$ is the database with the temporal modules $\overline{M}_1, \ldots, \overline{M}_m$ and each $\overline{M}_i$ is the minimal closure of M_i under the assumptions , $_{M_i}$.

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 using semantic assumptions in answering queries. However, as mentioned in the introduction, 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 , , a query Q' is said to be a , -*captured version of* Q if $Q'[DB] = Q[DB, ,]$ for all instantiations of 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 , -captured version, we limit the set of assumptions to those that can be expressed in flat-MQL^F formulas.[3] In particular, for each temporal module scheme $\mathtt{M} = (R, \mu)$ and assumption $P = P_X(Y_1^{meth_1} \ldots Y_k^{meth_k})$ with $R = XY_1 \ldots Y_k$ 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 predicate $\mathtt{M}$ has arity $\|R\| + 1$ and (b) for each module M on $\mathtt{M}$, $f_P(M) = \{x_1, \ldots, x_n, t \mid \Psi_P^{\mathtt{M}}\}$. It is clear that by identifying $\Psi_P^{\mathtt{M}}$ for each temporal scheme $\mathtt{M}$, we have identified the mapping f_P.

In the remainder of this section, we first show how assumption methods can be described by syntactic expressions from which we can derive the Ψ formulas corresponding to the assumptions. In particular, we use a slight extension of flat-MQL^F to syntactically specify assumption methods, and then we give a procedure on how to derive the aforementioned Ψ formulas for semantic assumptions from these syntactic specifications.

The syntactic specification of assumption methods need to be parametric with respect to the sets of attributes X and Y and to the module scheme $\mathtt{M}$. For this purpose we extend flat-MQL^F as follows: The number n of data attributes of the predicate $\mathtt{M}$ (and corresponding variables) is left as a parameter, and the symbol $\mathrm{Ind}_V^{\mathtt{M}}$, where V is a set of attributes, can be used to indicate the set of indices (positions) of attributes in V appearing in $\mathtt{M}$ when the "parameterized" scheme is instantiated.[4] We call a formula in this extension a *formula template*.

In this paper, we require that all assumption methods be specified by such an extension of flat-MQL^F. Such a requirement is not overly restrictive. Indeed, flat-MQL^F is a very expressive language, and we can specify many methods that are useful in practice. As an example, the formula template for the persistence method is as follows:

$$\begin{aligned}\Psi^{\mathtt{M}}_{P_V(W^{persis})}(w_1, \ldots, w_n, t) = \mathtt{M}(w_1, \ldots, w_n, t) \vee \\ (\exists t', v_1, \ldots, v_n) \quad (t' < t \ \wedge \ \mathtt{M}(v_1, \ldots, v_n, t') \wedge \\ (\forall t'', z_1, \ldots, z_n)(t' < t'' \leq t \Rightarrow \neg(\mathtt{M}(z_1, \ldots, z_n, t'') \ \wedge \ (\forall i \in \mathrm{Ind}_V^{\mathtt{M}})\, z_i = v_i)) \wedge \\ (\forall i \in \mathrm{Ind}_{VW}^{\mathtt{M}})\, w_i = v_i)\end{aligned}$$

We shall see below how the above template is used to derive the required formula $\Psi_P^{\mathtt{M}}$ where P is a

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

[4] For example, assume $\mathtt{M} = (\langle A, B, C, D, E\rangle, \mu)$ and $V = \{B, C, E\}$. Then $\mathrm{Ind}_V^{\mathtt{M}} = \{2, 3, 5\}$.

persistence assumption and $\mathtt{M}$ is a temporal module scheme. However, it is quite clear that the above formula template corresponds to our intuitive definition of persistence.

As another example, we formally define the *avg* assumption method by giving the following formula template:

$$\begin{aligned}
&\Psi^{\mathtt{M}}_{P_V(W^{avg})}(w_1,\ldots,w_n,t) = \mathtt{M}(w_1,\ldots,w_n,t) \vee \\
&\quad (\exists t_1,t_2,v_1,\ldots,v_n,u_1,\ldots,u_n)\,(t_1 < t \ \wedge\ \mathtt{M}(v_1,\ldots,v_n,t_1) \\
&\qquad \wedge\ (\forall t'',z_1,\ldots,z_n)\quad (t_1 < t'' \leq t \Rightarrow \neg(\mathtt{M}(z_1,\ldots,z_n,t'')\ \wedge\ (\forall i \in \mathrm{Ind}^{\mathtt{M}}_V)\, z_i = v_i)) \\
&\qquad \wedge\ t < t_2\ \wedge\ \mathtt{M}(u_1,\ldots,u_n,t_2) \\
&\qquad \wedge\ (\forall t'',z_1,\ldots,z_n)\quad (t \leq t'' < t_2 \Rightarrow \neg(\mathtt{M}(z_1,\ldots,z_n,t'')\ \wedge\ (\forall i \in \mathrm{Ind}^{\mathtt{M}}_V)\, z_i = u_i)) \\
&\qquad \wedge\ (\forall i \in \mathrm{Ind}^{\mathtt{M}}_V)\, w_i = v_i = u_i \\
&\qquad \wedge\ (\forall i \in \mathrm{Ind}^{\mathtt{M}}_W)\, w_i = \tfrac{v_i+u_i}{2})
\end{aligned}$$

It is easily seen that the mapping given by the above template correspond to the intuition behind the average method.

We now provide a procedure to obtain the formula $\Psi^{\mathtt{M}}_P$ for each given temporal scheme $\mathtt{M}$ and assumption P provided that each assumption method appearing in P is specified by a formula template.

Transformation 1 Given the point-base assumption $P = P_X(Y_1^{meth_1}\ldots Y_k^{meth_k})$ and temporal module scheme $\mathtt{M} = (R,\mu)$, the following steps output the formula $\Psi^{\mathtt{M}}_P(x_1,\ldots,x_n,t)$:

1. Derive the formulas $\Psi^{\mathtt{M}}_{P_X(Y_1^{meth_1})}(w_1^1,\ldots,w_n^1,t),\ldots,\Psi^{\mathtt{M}}_{P_X(Y_k^{meth_k})}(w_1^k,\ldots,w_n^k,t)$ from the formula template of methods as follows:

   - the free variable symbols are substituted with the ones given here;
   - V and W are instantiated with X and Y and μ with the given value of μ;
   - the number of variables n is set to $\|R\|$;
   - each expression of the form "$(\forall i \in \mathrm{Ind}^{\mathtt{M}}_Z)\ v_i = \ldots = w_i$" obtained after instantiating V with X and W with Y and possibly renaming the variables, is written out as a conjunction of a finite number of $v_l = \ldots = w_l$ for each l in $\mathrm{Ind}^{\mathtt{M}}_Z$.

2. Output the following formula as $\Psi^{\mathtt{M}}_{P_X(Y_1^{meth_1}\ldots Y_k^{meth_k})}(x_1,\ldots,x_n,t)$:

$$\begin{aligned}
&\exists w_1^1,\ldots,w_n^1\ \Psi^{\mathtt{M}}_{P_X(Y_1^{meth_1})}(w_1^1,\ldots,w_n^1,t)\ \wedge\ \ldots\ \wedge\ \exists w_1^k,\ldots,w_n^k\ \Psi^{\mathtt{M}}_{P_X(Y_k^{meth_k})}(w_1^k,\ldots,w_n^k,t)\ \wedge \\
&\quad (\forall i \in \mathrm{Ind}^{\mathtt{M}}_X)\,(x_i = w_i^1 = \ldots = w_i^k)\ \wedge (\forall i \in \mathrm{Ind}^{\mathtt{M}}_{Y_1})\,(x_i = w_i^1)\ \wedge\ \ldots\ \wedge\ (\forall i \in \mathrm{Ind}^{\mathtt{M}}_{Y_k})\,(x_i = w_i^k)
\end{aligned}$$

It is easily seen that the above formula is equivalent to the natural join operation required in the definition of assumption semantics. Hence, the following holds:

Proposition 3 Transformation 1 correctly identifies the formula $\Psi^{\mathtt{M}}_P$ for a point-based assumption P and temporal module scheme $\mathtt{M}$.

As an example, consider the mapping for $P = P_{AC}((BD)^{avg}\,(EF)^{persis})$. Given a module M on scheme $\mathtt{M}$=$(\langle A,B,C,D,E,F\rangle,\mu)$, this assumption intuitively says that values for attributes B and D should be derived with respect to AC using the avg (average) method while values of attributes E and F should simply persist (with respect to AC). In this example $n = 6$ and $\mathrm{Ind}^{\mathtt{M}}_{AC} = \{1,3\}$, $\mathrm{Ind}^{\mathtt{M}}_{BD} = \{2,4\}$,

and $\mathrm{Ind}^{\mathtt{M}}_{EF} = \{5,6\}$. The following is the formula defining $\Psi^{\mathtt{M}}_{P}$. (In the formula, the temporal variable t is of sort μ.)

$$\begin{aligned}\Psi^{\mathtt{M}}_{P}(x_1,\ldots,x_6,t) \;=\; & \exists w^1_1,\ldots,w^1_6\ \Psi^{\mathtt{M}}_{P_{AC}(BD^{avg})}(w^1_1,\ldots,w^1_6,t)\ \wedge \\ & \exists w^2_1,\ldots,w^2_6\ \Psi^{\mathtt{M}}_{P_{AC}(EF^{persis})}(w^2_1,\ldots,w^2_6,t)\ \wedge \\ & x_1 = w^1_1 = w^2_1\ \wedge\ x_3 = w^1_3 = w^2_3\ \wedge \\ & x_2 = w^1_2\ \wedge\ x_4 = w^1_4\ \wedge\ x_5 = w^2_5\ \wedge\ x_6 = w^2_6\end{aligned}$$

where

$$\begin{aligned}\Psi^{\mathtt{M}}_{P_{AC}(EF^{persis})}(w^2_1,\ldots,w^2_6,t) = \mathtt{M}(w^2_1,\ldots,w^2_6,t)\ \vee\ (\exists t')\quad & (t' < t\ \wedge\ \mathtt{M}(w^2_1,\ldots,w^2_6,t') \\ \wedge\ (\forall t'', z_1,\ldots,z_6)(t' < t'' \leq t \Rightarrow \neg(M(z_1,\ldots,z_6,t'')\ \wedge\ & z_1 = w^2_1\ \wedge\ z_3 = w^2_3)))\end{aligned}$$

and $\Psi^{\mathtt{M}}_{P_{AC}(BD^{avg})}(w^1_1,\ldots,w^1_6,t)$ is similarly derived.

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

Algorithm 1 Let Q be a correctly-typed MQL^F query and , 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 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, ..., P_k are all the assumptions in , $\mathtt{M}$. □

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 2 The query obtained by Algorithm 1 is a , -captured version of Q.

Example 3 Let us consider the flat-MQL^F query given earlier:
$Q = \{x,t \mid \exists y,z,w\,(\mathtt{ACCOUNTS}(x,y,z,w,t) \wedge z < 1000)\}$. If $DB = \{\mathtt{ACCOUNTS}\}$ and , $= \{P_1\}$ 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 that specifies the semantics of the persistence method, we can easily verify that the answer $Q'[DB]$ is the module $(\mathtt{AcctNo},\mathtt{second},\varphi)$ 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 by 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. □

5 Interval-Based Assumptions

Interval-based assumptions are 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 the value is being derived.

5.1 An example: Liquidity

With $I_X(A^{\downarrow})$ we denote the assumption of the attribute A being *downward hereditary* with respect to the attributes in X. This intuitively means that if we have an explicit value for the attribute A with respect to certain values for X 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 with respect to the same values for X. Referring to our running example, it is reasonable to assume $I_{\texttt{AcctNo}}(\texttt{AccHol}^{\downarrow})$; intuitively, if an account (`AcctNo`) has a corresponding account holder name (`AccHol`) in a second, then it has the same account holder name in any millisecond, microsecond, and so on within that second. Similarly, with $I_X(A^{\uparrow})$ we denote the assumption of the attribute A being *upward hereditary* with respect to X. Intuitively, if we have the same value for the attribute A with respect to the same X at contiguous ticks, that value is also the value of A for the same X 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 $I_X(A^{\updownarrow})$ we denote the assumption of the attribute A being *liquid* with respect to X; i.e., it is both downward and upward hereditary. Referring to our example, we can reasonably make the assumption $I_{\texttt{AcctNo}}((\texttt{AccHol}, \texttt{AnIntRate})^{\updownarrow})$.

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

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

This query is not correctly typed since it asks for the account holder name of account 1001 in months, while the temporal type of `ACCOUNTS` is in seconds. However, we can answer the query if there are assumptions, such as the upward heredity assumption, for all the attributes in `ACCOUNTS` to convert the information to a coarser type. Note that the example query could be used to see if there has been a change of name during a month, in which case the answer would be empty.

We now formalize the notion of a general interval-based assumption.

5.2 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 that there is a fixed set of conversion methods.

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

In Subsection 5.1 we have illustrated three conversion methods. There are many others that can be useful. As an example, consider a module of type μ and assume that i is a tick of a type ν such that there is a sequence of ticks $j_1, \ldots, j_k$ of type μ, all contained in tick i of ν. We can define two conversion methods—we call them *first* and *last*, respectively—that derive the value for tick i of ν using the values at tick j_1 and j_k of μ, respectively. In our running example, it is reasonable to make the assumption that *last* can be used to convert the `Balance` attribute with respect to the attribute `AcctNo`. This means that if a query on the `ACCOUNTS` module asks for the balance of an account for a month, the balance at the last second of that month for that account is returned. Note that even though we do not have explicit values for the last second of each month, we can still derive them using the persistence assumption. Similarly, considering all other attributes in `ACCOUNTS`, the following is a reasonable interval-based assumption: $I_{\texttt{AcctNo}}((\texttt{AccHol}, \texttt{AnIntRate})^{\updownarrow}, \texttt{Balance}^{last})$.

Analogous to point-based assumptions, we define the semantics of conversion methods before giving the semantics of interval-based semantic assumptions. Given a temporal module $M = (R, \mu, \varphi)$ and a set S of integers, we use the notation $M|_S$ to denote the temporal module (R, μ, φ') where φ' is defined as follows: For each positive integer i, $\varphi'(i) = \varphi(i)$ if $\mu(i) \cap S \neq \emptyset$, and $\varphi'(i) = \emptyset$ otherwise. That is, $M|_S$ is a temporal module obtained by keeping only the values of the windowing function at the ticks which intersect with the absolute time set S.

Definition (Semantics of conversion methods)
The semantics of a conversion method *conv* is given by a mapping $conv()$ from two sets of attributes, X and Y, a temporal module $M = (R, \mu, \varphi)$ with $XY \subseteq R$, and a temporal type ν such that the following conditions are satisfied:

1. $conv(X, Y, M, \nu)$ is a temporal module on (XY, ν);
2. $conv(X, Y, M, \mu) = \pi^T_{XY}(M)$;
3. for each subset Z of Y, $conv(X, Z, \pi^T_{XZ}(M), \nu) = \pi^T_{XZ} conv(X, Y, M, \nu)$;
4. for each value x of X, $conv(X, Y, \sigma^T_{X=x}(M), \nu) = \sigma^T_{X=x}(conv(X, Y, M, \nu))$; and
5. for each positive integer j, $conv(X, Y, M|_{\nu(j)}, \nu) = conv(X, Y, M, \nu)|_{\nu(j)}$.

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 coincides with the temporal type of the input module, then the output module should simply be a projection of the input module. The third condition requires that each method derives values for a certain attribute independently from values of other attributes. The fourth condition requires that each method derives values for a certain attribute considering only tuples in the original module having the same value for X. The final condition says that the value at each tick j in the target type ν depends only on values at ticks that intersect with j.

If a module $M = (R, \mu, \varphi)$ has a non-empty set of associated interval-based assumptions, values for modules in different temporal types can be implied by these assumptions. The way of deriving these values is specified by the semantics of the semantic assumptions:

Definition (Semantics of interval-based assumptions)
For each interval-based assumption $I = I_X(Y_1^{conv_1} \ldots Y_k^{conv_k})$, there is an associated (partial) mapping f_I from temporal modules to temporal modules such that, for each temporal module $M = (R, \mu, \varphi)$ with $XY_1 \ldots Y_k \subseteq R$, if $conv_j(X, Y_j, M, \nu) = (XY_j, \nu, \varphi_j)$ for each $j = 1, \ldots, k$, then $f_I(M, \nu) = (XY_1 \ldots Y_k, \nu, \varphi')$ where $\varphi'(i) = \varphi_1(i) \bowtie \ldots \bowtie \varphi_k(i)$ for each $i \geq 1$.

Hence, analogous to point-based assumptions, the values that are derived according to different conversion methods within one assumption are combined via the natural join. Notice that the mapping for an interval-based assumption defined this way satisfies all the conditions required for the mappings of conversion methods.

6 Query Evaluation with Interval-Based Assumptions

The notion of query answering with point-based assumptions in Section 4 can be extended to that with interval-based assumptions. As in Section 4, we assume there is a set , $_{\mathtt{M}}$ of interval-based assumptions for each $\mathtt{M}$ in the database, and denote the collection of all interval-based assumptions by , .

As for point-based assumptions, we restrict ourselves to assumptions involving all the attributes appearing in the modules on which they are applied.[5] We also limit interval-based assumptions for which we can evaluate queries to those that can be expressed in MQL^F formulas. In particular, for each temporal module scheme $\mathtt{M} = (R, \mu)$, temporal type ν and assumption $I = I_X(Y_1^{conv_1} \ldots Y_k^{conv_k})$ with $R = XY_1 \ldots Y_k$ we require the existence of a formula $\Psi_I^{\mathtt{M},\nu}$ in MQL^F such that (a) among the predicates in the formula, only one predicate symbol $\mathtt{M}$, involving both data and temporal sorts, appears (possibly several times) in $\Psi_I^{\mathtt{M},\nu}$ and predicate $\mathtt{M}$ has arity $\|R\| + 1$, and (b) if M is a module on $\mathtt{M}$ then $f_I(M, \nu) = \{x_1, \ldots, x_n, t{:}\nu \mid \Psi_I^{\mathtt{M},\nu}\}$ where $n = \|R\|$.

Analogous to assumption methods, conversion methods can be defined through formula templates such that the Ψ formulas of assumption involving those method can be easily derived from the templates. The formula template needs to be parametric with respect to the module scheme (R, μ), the target type ν and to the sets of attributes X and Y. For this purpose the same simple extension considered for flat-MQL^F can be used here, allowing n to remain a parameter, and the symbol $\mathrm{Ind}_V^{\mathtt{M}}$ where V is a set of attributes can be used to indicate the set of indices (positions) of attributes in V appearing in R when the scheme is instantiated. The restriction to this language for the definition of conversion methods is reasonable, since MQL^F can be used to express many practically interesting interval-based assumptions. For example, the liquidity conversion method is specified by the following formula template:

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

We shall see below how the above template is used to derive the required formula $\Psi_I^{\mathtt{M},\nu}$, where I is a liquidity assumption of the form $I_X(Y^{\updownarrow})$. However, it should be intuitively clear that this formula template gives the mappings for the liquidity conversion.

The conversion *last* can be specified by the formula template:

$$\begin{aligned}\Psi^{\mathtt{M},\nu}_{I_V(W^{last})}(x_1, \ldots, x_n, t) = \\ (\exists t{:}\mu)(\mathbf{IntSec}_{\mu,\nu}(t', t) \ &\wedge (\forall t'' : \mu)\ (t'' > t' \Rightarrow \neg \mathbf{IntSec}_{\mu,\nu}(t'', t)) \\ &\wedge (\forall s{:}\nu)\ (s > t \Rightarrow \neg \mathbf{IntSec}_{\mu,\nu}(t', s)) \\ &\wedge (\exists w_1, \ldots, w_n)\ (\mathtt{M}(w_1, \ldots, w_n, t') \ \wedge \ (\forall j \in \mathrm{Ind}_{VW}^{\mathtt{M}})\ x_j = w_j))\end{aligned}$$

Once we have the definition of conversion methods in the form of the formula templates, for any semantic assumption I involving the defined methods, we can easily obtain the formula $\Psi_I^{\mathtt{M},\nu}$ such that $f_I(M, \nu) = \{x_1, \ldots, x_n, t{:}\mu \mid \Psi_I^{\mathtt{M},\nu}\}$ for each temporal module M on $\mathtt{M}$ and type ν.

Similar to what we have done for point based assumptions, the formula $\Psi_I^{\mathtt{M},\nu}$ for an assumption $I = I_X(Y_1^{conv_1} \ldots Y_k^{conv_k})$, a target type ν, and a temporal module scheme $\mathtt{M} = (R, \mu)$ such that $XY_1 \ldots Y_k = R$ with $\|R\| = n$ can be obtained by the following transformation.

Transformation 2 Given an interval based assumption $I_X(Y_1^{conv_1} \ldots Y_k^{conv_k})$, temporal module scheme $\mathtt{M}$ and (target) temporal type ν, the following steps output the formula $\Psi_I^{\mathtt{M},\nu}(x_1, \ldots, x_n, t)$:

1. Derive the formulas $\Psi^{\mathtt{M},\nu}_{I_X(Y_1^{conv_1})}(w_1^1, \ldots, w_n^1, t), \ldots, \Psi^{\mathtt{M},\nu}_{I_X(Y_k^{conv_k})}(w_1^k, \ldots, w_n^k, t)$ from the semantics of conversion methods as follows:
   - the free variable symbols are substituted with the ones given here;
   - V and W are instantiated with X and Y, while μ and ν with their given values;
   - the number of variables n is set to $\|R\|$;

[5] A notion of minimal closure analogous to that for point-based assumptions can be easily defined.

- each expression of the form "$(\forall i \in \mathrm{Ind}_Z^{\mathtt{M}})\ v_i = \ldots = w_i$" obtained after instantiating V and W with X and Y and possibly renaming the variables, is written out as a conjunction of a finite number of $v_l = \ldots = w_l$ for each l in $\mathrm{Ind}_Z^{\mathtt{M}}$.

2. Output the following as $\Psi_I^{\mathtt{M},\nu}(x_1, \ldots, x_n, t)$:

$$\exists w_1^1, \ldots, w_n^1\ \Psi^{\mathtt{M},\nu}_{I_X(Y_1^{conv_1})}(w_1^1, \ldots, w_n^1, t)\ \wedge\ \ldots\ \wedge\ \exists w_1^k, \ldots, w_n^k\ \Psi^{\mathtt{M},\nu}_{I_X(Y_k^{conv_k})}(w_1^k, \ldots, w_n^k, t)\ \wedge$$
$$(\forall i \in \mathrm{Ind}_X^{\mathtt{M}})\,(x_i = w_i^1 = \ldots = w_i^k)\ \wedge (\forall i \in \mathrm{Ind}_{Y_1}^{\mathtt{M}})\,(x_i = w_i^1)\ \wedge\ \ldots\ \wedge\ (\forall i \in \mathrm{Ind}_{Y_k}^{\mathtt{M}})\,(x_i = w_i^k)$$

It is easily seen that the above formula is the equivalent in our logic of the natural join operation required in the definition of interval-based assumption semantics. Hence, the following holds:

Proposition 4 Transformation 2 identifies the mapping for an interval-based assumption.

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.

However, there are cases where assumptions are not "designed" for certain information derivation. For example, if only downward heredity is assumed on a temporal module `M`, information in terms of any temporal type that is coarser than $\mu_{\mathtt{M}}$ is not derivable. Such incorrectly typed queries cannot be answered. Below we formalize this intuition.

Definition (Type reachability)
Let I be an interval-based assumption and $\Psi_I^{\mathtt{M},\nu}$ is its associated MQL^F formula. Suppose μ and ν are two temporal types. Then a *type μ can reach type ν via I* if there exists a temporal module $\mathtt{M} = (R, \mu, \varphi)$ such that the answer of the query $\{x_1, \ldots, x_n, t{:}\nu | \Psi_I^{\mathtt{M},\nu}\}$, where x_1, ..., x_n are all the free variables in $\Psi_I^{\mathtt{M},\nu}$, is not empty.

In other words, type μ cannot reach type ν via I if the answer of the query given in the definition is empty independently from the module M. Thus, no matter what is the given information (which is in terms of μ), no information in terms of ν can be derived by using I.

We can now define the notion of *doable queries*, which are incorrectly typed queries that can be answered in the presence of interval-based assumptions.

Definition (Doable query)
We say that a query is *doable* (wrt the interval-based assumptions ,) if for each subformula $\mathtt{M}(x_1, \ldots, x_n, t)$ appearing in the query, where t is of type ν, ν can be reached from $\mu_{\mathtt{M}}$ via some interval-based assumptions in , $\mathtt{M}$.

The doability of queries is decidable if all conversion methods are specified by MQL^F formula templates that use no arithmetic operations and with the assumption that the **IntSec** predicate is precomputed into a finite table with a constant lookup time. Unfortunately, the complexity of the decision procedure even under these simplifying assumptions is rather high.

Proposition 5 Let , be a set of interval-based assumptions whose conversion methods are specified by MQL^F formula template with the above simplification and assumption. Then, the problem of determining if a query is doable under , is decidable, but the complexity of such a decision procedure is at least $2^{2^{cn}}$ where n is the length of the query and c is a constant.

In terms of practical systems, we require that the system designer gives explicit information about interval-based assumptions. We identify below two main kinds of interval-based assumptions: upward and downward assumptions.

Definition We say that an assumption I is *upward* (*downward*, resp.) if for each type ν coarser (finer, resp.) than μ, type ν can be reached from μ via I.

Clearly, the liquidity assumption is both upward and downward. It is easy to define in MQL^F the two variants of liquidity so that one works as *upward heredity* and the other as *downward heredity*. Each one has only one of the two properties defined above. If our interval-based assumptions are all upward and/or downward, doability is easily checked.

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

Definition Let , be a set of interval-based assumptions and Q a MQL^F query wrt , . 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 $\overline{\mathtt{M}}_{i,\nu} = (R_i, \mu_i, \varphi_i)$, 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* , (denoted by $Q[DB, ,]$) 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 , *-captured version of* Q if $Q'[DB] = Q[DB, ,]$.

That is, to answer a query with assumptions , , 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 , -captured version is a query which will obtain, without changing the database, the correct answer of the original query with assumptions.

Algorithm 2 Given an 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 , $\mathtt{M}$. □

Theorem 3 Given a query Q and a set of interval-based assumptions , as inputs to Algorithm 2, the query obtained by the algorithm is a , -captured version of Q.

7 Combining Point-Based and Interval-Based Assumptions

If both point-based and interval-based assumptions are present in , $\mathtt{M}$, 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 formal definition of answer can be given as follows:
$Q[DB, , _I \cup , _P] = \overline{Q}[\overline{DB}]$ where

- $\overline{DB} = \{\overline{\mathtt{M}}_{i,\nu} \mid \overline{\mathtt{M}}_{i,\nu} = \bigcup^{\mathrm{T}}_{I \in \Gamma_{\mathtt{M}_i}} f_I(\overline{\mathtt{M}}_i, \nu), \overline{\mathtt{M}}_i$ is the minimal closure of $\mathtt{M}_i \in DB$ 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.

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 4 Let , be a set of point-based and/or interval-based assumptions and Q an MQL^F query. Then, Algorithm 1 and 2, applied in this order, yield a , -captured version of Q.

Example 4 Let DB={ACCOUNTS, RISKS}, where ACCOUNTS and RISKS are temporal modules of Exampls 1 and 2 respectively, , $= \{P_1, P_2, I_1, I_2\}$ where
$P_1 = P_{\mathtt{AcctNo}}((\mathtt{AccHol}, \mathtt{Balance}, \mathtt{AnIntRate})^{persis})$,
$P_2 = P_{\mathtt{AccHol}}(\mathtt{EstRisk}^{avg})$,
$I_1 = I_{\mathtt{AcctNo}}((\mathtt{AccHol}, \mathtt{AnIntRate})^{\updownarrow}, \mathtt{Balance}^{last})$, and
$I_2 = I_{\mathtt{AccHol}}(\mathtt{EstRisk}^{\updownarrow})$.
Consider the query

$$Q = \{x, t : \mathtt{month} \mid \exists y, z, w, r\,(\mathtt{ACCOUNTS}(x, y, z, w, t) \wedge z < 1000 \wedge \mathtt{RISKS}(y, r, t) \wedge r > 3)\}$$

This query asks for accounts having a balance under \$1,000 and such that the estimated risk for their account holder is greater than 3. The query asks for these account numbers in terms of months. This means that only the accounts such that those conditions are verified on an interval corresponding to a month would be returned in the answer. The answer $Q[DB, ,]$ can be obtained applying first the point-based assumption P_1 and P_2 respectively on the two modules in DB. Then, applying the interval-based assumption I_1 and I_2 respectively to the resulting modules, they will be converted in terms of type month. The application of P_1 and P_2 ensures that there will be at least one tuple respectively for each second in ACCOUNTS and for each day in RISKS. About interval-based assumptions, since Balance is converted using *last* and the other attributes in ACCOUNTS are liquid, the resulting MONTHLY-ACCOUNTS has the following windowing function:
$\varphi(i) = \{(1001, \mathtt{J.Smith}, 3500, 4.00), (1500, \mathtt{A.Brady}, 1500, 3.00), (2034, \mathtt{T.Ford}, 500, 2.50)\}$
for April of 1993$\leq i \leq$ June of 1993 and
$\varphi(i) = \{(1001, \mathtt{J.Smith}, 4000, 4.00), (1500, \mathtt{A.Brady}, 1500, 3.00), (2034, \mathtt{T.Ford}, 500, 2.50)\}$
for $i \geq$ July of 1993. Similarly, the resulting MONTHLY-RISKS has windowing function
$\varphi'(j) = \{(\mathtt{Smith}, 3), (\mathtt{Ford}, 3.5)\}$ for March of 1993$\leq j \leq$ May of 1993, and empty for other ticks.

The answer can now be computed evaluating Q on this new database and in this case it is a module whose windowing function has the only value 2034 for ticks corresponding to April and May of 1993. Theorem 4 says that we don't need to compute this new database, but we can simply modify the query, yielding the same answer. □

8 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, an 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 , of assumptions, a query Q is *embedded domain independent wrt* , if the (non-temporal) values in the answer of Q with assumptions , 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.[6] An 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, ..., y_m are all limited.

An 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. This in particular means that if the set of data (non-temporal) values appearing in the database is finite, then the set of (non-temporal) values appearing in the answer of a safe query is also finite.

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 $\Psi_P^{\mathtt{M}}$ is safe for any temporal module scheme $\mathtt{M}$ (resp. $\Psi_I^{\mathtt{M},\nu}$ is safe for any temporal module scheme $\mathtt{M}$ and type ν).

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

The following proposition formally states that the condition on the safety of assumptions is sufficient.

Proposition 6 If Q is a safe query and , a set of safe assumptions, then Q is embedded domain independent with respect to , .

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

We conclude, by the following proposition, that to ensure the embedded domain independence of the query that we are evaluating, it is sufficient to check the safety of the original query if all the assumptions are assumed to be safe.

Proposition 7 Let Q be a query and , a set of assumptions. Suppose each assumption in , , as well as Q, is safe. Then Q', obtained by the transformation from Q according to , , is embedded domain independent.

Proof. By Theorem 4, $Q'[DB] = Q[DB, ,]$. By Proposition 6, Q is embedded domain independent wrt , . Hence, Q' is embedded domain independent. □

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_{max}, 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, a temporal module $M = (R, \mu, \varphi)$ 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 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 the predicates **IntSec** and $<$.

Theorem 5 The answer to an embedded-domain independent 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.

We now show that the flat-MQL^F language enjoys a better property regarding query answers.

Theorem 6 Let , be a set of assumptions expressed in terms of flat-MQL^F formulas. The answer to an embedded-domain independent query in flat-MQL^F on a database with assumptions , 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 , . Indeed, the whole process described in the proof of Theorem 6 is a syntactical process from which t_{max} can be derived.

To conclude this section, we mention that Theorem 6 does not hold for MQL^F queries. Consider a temporal module $(R, \mathtt{day}, \varphi)$, where $R = \langle \mathtt{NameOnDuty} \rangle$, which specifies the person on duty for each day. The persistence assumption is assumed on the module, i.e., if no one is specified to be on duty on a specific day, the last person on duty should continue. Now consider the following query:

$$\{x, i : \mathtt{day} | \mathtt{M}(x, i) \wedge \exists j : \mathtt{month}(\mathbf{IntSec}_{\mathtt{month},\mathtt{day}}(j, i) \wedge \neg \exists i_2 : \mathtt{day}(\mathbf{IntSec}_{\mathtt{month},\mathtt{day}}(j, i_2) \wedge i_2 < i))\}.$$

This query asks the list of persons (and the days) who are on duty on first day of month. Clearly, the result module is usually not eventually uniform. Indeed, the result module alternates a non-empty tick (i.e., $\varphi(i) \neq \emptyset$) with a sequence of empty ticks (i.e., $\varphi(i) = \emptyset$), up to the infinite.

From the results of this section, we conclude that when a temporal database involves only one temporal type, then we need only eventually uniform temporal modules. This justifies the choice researchers have made in most temporal database literature. On the other hand, if more than one types are involved, then eventual uniformity does not seem to be enough. We may have to resort to first-order finite partitioning. Ultimate periodicity (such as [KSW90]) may be a good candidate as a special case of first-order finite partitioning.

9 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.

A temporal database with a set of assumptions is equivalent to a larger (sometimes infinite) database where values (implied by assumptions) are made explicit. We defined as the minimal closure of the database the minimal of these databases with explicit values. If each assumption involves all the attributes of the scheme to which it is applied, then this minimal closure 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. For example, the query language could allow to talk about part of a temporal module: if $\mathtt{M}$ is a temporal module on a scheme with three attributes, the formula could contain $\mathtt{M}(x, *, y, t)$, where $*$ is a don't care. In this case, the "don't care" may not have a value. This can be a straightforward extension of our work.

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. Consider a *summation* conversion method deriving an attribute value at a tick of time as the sum of the values at ticks of finer temporal types contained in it. It is impossible to express this method without having a bound on the

number of ticks contained in each coarser tick. Hence, to represent this kind of methods we require an additional constraint on the set of types used in the database:

> Given an arbitrary type μ in the set, it exists a number k (bound) such that each tick of μ contains at most k ticks of each other type in the set.

In the case of *summation* this condition ensures that there is a bound to the number of values that will be summed. Observe that the above requirement is satisfied for many sets of types of practical utility.

There are other 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 assumptions, 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.

References

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

[CCT94] James Clifford, Albert Croker, and Alexander Tuzhilin. On completeness of historical relational query languages. *ACM Transactions on Database Systems*, 19(1):64–116, March 1994.

[Cho92] J. Chomicki. History-less checking of dynamic integrity constraints. In F. Golshani, editor, *Proceedings of the International Conference on Data Engineering*, volume 8, pages 557–564, Los Alamitos, CA, February 1992. IEEE Computer Society Press.

[Cho94] Jan Chomicki. Temporal query languages: A survey. In D.M.Gabbay and H.J. Ohlbach, editors, *Temporal Logic, First International Conf.*, volume 827 of *LNAI*, pages 506–534, Bonn, Germany, 1994. Springer-Verlag.

[CI94] J. Clifford and T. Isakowitz. On the semantics of (bi)temporal variable databases. In M. Jarke, J. Bubenko, and K. Jeffery, editors, *Proceedings of 4th International Conference on Extending Database Technology*, pages 215–230, March 1994.

[Coo72] C.D. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7:91–99, 1972.

[CT85] J. Clifford and A.U. Tansel. On an algebra for historical relational databases: Two views. In S. Navathe, editor, *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 247–265, Austin, TX, May 1985. ACM.

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

[DM87] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence Journal*, 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 Symp. on Principles of Database Systems*, pages 253–264, Washington, DC, May 1993.

[FR79] J. Ferrante and C. W. Rackoff. *The computational complexity of logical theories.* Number 718 in Lecture Notes in Mathematics. Spinger-Verlag, 1979.

[GHR94] D. M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical foundations and computational aspects*, volume 1 of *Oxford Science Publications.* Clarendon Press, Oxford, 1994.

[JSS93] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying temporal data models via a conceptual model. Technical Report TR 93-31, Department of Computer Science, University of Arizona, Tucson, AZ, 1993.

[KSW90] F. Kabanza, J-M. Stevenne, and P. Wolper. Handling infinite temporal data. In *Proc. ACM Symp. on Principles of Database Systems*, Nashiville, Tennessee, April 1990. ACM.

[NS92] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *First International Conference on Information and Knowledge Management*, Baltimore, MD, November 1992.

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

[Sno84] R. Snodgrass. The temporal query language TQuel. In *Proc. ACM Symp. on Principles of Database Systems*, pages 204–212, Waterloo, Ontario, Canada, April 1984.

[SS87] A. Segev and A. Shoshani. Logical modeling of temporal data. In U. Dayal and I. Traiger, editors, *Proceedings of the ACM SIGMOD Annual Conference on Management of Data*, pages 454–466, San Francisco, CA, May 1987. ACM, ACM Press.

[Tan87] A.U. Tansel. A statistical interface for historical relational databases. In *Proceedings of the International Conference on Data Engineering*, pages 538–546, Los Angeles, CA, February 1987. IEEE Computer Society, IEEE Computer Society Press.

[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, ISSE Department, George Mason University, 1994.

[WJL91] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.

[WJS93] X. Wang, S. Jajodia, and V.S. Subrahmanian. Temporal modules: An approach toward federated temporal data bases. In *Proceedings of 1993 ACM SIGMOD International Conference on the Management of Data*, Washington, D.C., 1993.

Appendix

A.1 Proof of Proposition 1

Proof. Let M be a temporal module on (R, μ). Since each assumption P in , involves all the attributes in R, $f_P(M)$ is a temporal module on (R, μ) by definition. Assume $f_P(M) = (R, \mu, \varphi_P)$ for each P in , and let $\overline{M} = (R, \mu, \varphi)$, where φ is defined as follows: For each positive integer i, $\varphi(i) = \bigcup_{P \in \Gamma} \varphi_P(i)$. It is easily shown that $\overline{M}$ is a minimal closure of M under , . Indeed, it is clear that $\overline{M}$ is a closure of M under , . It is also clear that $\overline{M}$ is a minimal closure of M under , since, if a tuple t is dropped from $\varphi(i)$ for some i, then $\varphi_P(i) \not\subseteq \varphi(i)$ for some $P \in$, and M is no longer a closure. We now show that this minimal closure is unique. Suppose by contradiction that there exists a temporal module $M_1 = (R, \mu, \varphi_1)$ such that $M_1 \neq \overline{M}$ and M_1 is a minimal closure of M under , . By definition, $f_P(M) \preceq M_1$ for each $P \in$, . Hence, $\varphi_P(i) \subseteq \varphi_1(i)$ for each $P \in$, and $\bigcup_{P \in \Gamma} \varphi_P(i) \subseteq \varphi_1(i)$ for each positive integer i. By the definition of $\overline{M}$ above, it follows that $\varphi(i) \subseteq \varphi_1(i)$ and we conclude that $\overline{M} \preceq M_1$. This contradicts to the fact that $M_1 \neq \overline{M}$ and M_1 is a minimal closure of M under , . □

A.2 Proof of Proposition 2

Proof. Let $P_1 = P_X(Y_1^{meth_1} \ldots Y_k^{meth_k})$ and $P_2 = P_X(W_1^{meth_1} \ldots W_k^{meth_k})$ such that $W_j \subseteq Y_j$ for each $1 \leq j \leq k$. We show that $P_1 \models P_2$. Let $M = (R, \mu, \varphi)$ be a temporal module and $\overline{M} = (R, \mu, \overline{\varphi})$ be a closure of M under P_1. We only need to show that $\overline{M}$ is a closure of M under P_2. Since $\overline{M}$ is a closure of P_1, we have $f_{P_1}(M) \preceq \pi^T_{XY_1 \ldots Y_k}(\overline{M})$. Let $f_{P_1}(M) = (R, \mu, \varphi')$, and for each $1 \leq j \leq k$, let $meth_j(X, Y_j, M) = (XY_j, \mu, \varphi_j^Y)$. By definition, $\varphi_1^Y(i) \bowtie \cdots \bowtie \varphi_k^Y(i) = \varphi'(i)$ and $\varphi'(i) \subseteq \overline{\varphi}(i)$ for each positive integer i. Furthermore, let $f_{P_2}(M) = (R, \mu, \varphi'')$ and $meth_j(X, W_j, M) = (XW_j, \mu, \varphi_j^W)$ for each $1 \leq j \leq k$. By definition, $\varphi''(i) = \varphi_1^W(i) \bowtie \cdots \bowtie \varphi_k^W(i)$ for each positive integer i. Let i be a positive integer. To show that $\overline{M}$ is a closure of M under P_2, it suffices now to show that $\varphi''(i) \subseteq \pi_{XW_1 \ldots W_k}(\varphi(i))$. By definition again, $meth_j(X, W_j, M) = \pi^T_{XW_j}(meth_j(X, W_j, M)) = meth_j(X, W_j, \pi^T_{XW_j}(M)) = \pi^T_{XW_j}(meth_j(X, Y_j, M))$. Therefore, $\varphi_j^W(i) = \pi_{XW_j}(\varphi_j^Y(i))$. Hence, $\varphi''(i) = \pi_{XW_1}(\varphi_1^Y(i)) \bowtie \cdots \bowtie \pi_{XW_k}(\varphi_k^Y(i))$. Since $Y_1, \ldots, Y_k$ are pairwise disjoint, it is clear now that $\varphi''(i) = \pi_{XW_1 \ldots W_k}(\varphi_1^Y(i) \bowtie \cdots \bowtie \varphi_k^Y(i)) = \pi_{XW_1 \ldots W_k}(\varphi'(i)) \subseteq \pi_{XW_1 \ldots W_k}(\overline{\varphi}(i))$. □

A.3 Proof of Theorem 1

Proof.

Soundness

Let $Z \subseteq X$. We only need to prove that $P_Z(V^{persis})$ is implied by $P_X(Y^{persis})$ where $V = (X \setminus Z) \cup Y$. [Indeed, if this is the case, by Proposition 2, we know that $P_Z(W^{persis})$, where $W \subseteq V$, is implied by $P_X(Y^{persis})$.] In order to show this, let $M = (R, \mu, \varphi)$ be an arbitrary module and $\overline{M}$ a closure of M under $P_X(Y^{persis})$. Let $V = (X \setminus Z) \cup Y$. By definition, we have $f_{P_X(Y^{persis})}(M) \preceq \pi^T_{XY}(\overline{M})$, and we need to prove that $f_{P_Z(V^{persis})}(M) \preceq \pi^T_{ZV}(\overline{M})$. By the specification of the persistence semantics, $f_{P_Z(V^{persis})}(M) = (XY, \mu, \varphi')$ where, for each $i \geq 1$, $\varphi'(i) = \pi_{XY}\varphi(i) \cup \pi_{XY}S_Z$ where $S_Z = \{t \mid \exists j \quad j < i,\ t_j \in \varphi(j)$ and $\forall k, t_k \quad (j < k \leq i \ \wedge \ t_k \in \varphi(k)) \Rightarrow t_k[Z] \neq t_j[Z]\}$. Note that $f_{P_X(Y^{persis})}(M) = (XY, \mu, \varphi'')$ where, for each $i \geq 1$, $\varphi''(i) = \pi_{XY}\varphi(i) \cup \pi_{XY}S_X$ and S_X is defined as S_Z with X in place of Z. Since $Z \subseteq X$, it is easily seen that $t_k[Z] \neq t_j[Z]$ implies $t_k[X] \neq t_j[X]$ and hence, $S_Z \subseteq S_X$. Therefore, $f_{P_Z(V^{persis})}(M) \preceq f_{P_X(Y^{persis})}(M) \preceq \pi^T_{XY}(\overline{M})$.

Completeness

Let $P = P_Z(W^{persis})$ be a persistence assumption and , a set of persistence assumptions. Assume that for each $P_X(Y^{persis})$ in , , either $Z \not\subseteq X$, or $Z \subseteq X$ but $W \not\subseteq (X \setminus Z) \cup Y$. To prove the completeness, we only need to show that there exists a temporal module M such that a closure of M under , is not a closure of M under P. Let R be a relation scheme including all attributes appearing in the assumptions that we are considering, and μ an arbitrary type with at least two non-empty ticks. We define the module $M = (R, \mu, \varphi)$ with $\varphi(1) = \{(1, \ldots, 1)\}$, $\varphi(i) = \emptyset$ for each $i \geq 3$ and $\varphi(2)$ defined as follows: For each assumption $P_X(Y^{persis}) \in$, , if $Z \not\subseteq X$, $\varphi(2)$ includes the tuple t with $t[X] = 1$ and $t[R \setminus X] = 0$. For each assumption $P' = P_X(Y^{persis})$ in , , assume $f_{P'}(M) = (XY, \mu, \varphi_{P'})$. We now give a minimal closure of M under , . Let $\overline{M} = (R, \mu, \overline{\varphi})$ be the temporal module on (R, μ) whose windowing function $\overline{\varphi}$ is defined as follows: $\overline{\varphi}(1) = \varphi(1)$ and for each $i \geq 2$,

(*) $\quad \overline{\varphi}(i) = \varphi(i) \cup \{t' \mid \exists P' = P_X(Y^{persis}) \in , , \quad t'[XY] \in \varphi_{P'}(i) \quad \text{and } t'[R \setminus XY] = 0\}$.

It is easy to verify that $\overline{M}$ is a closure of M since $M \preceq \overline{M}$ and $f_{P'}(M) \preceq \pi_{XY}(\overline{M})$ for each $P' = P_X(Y^{persis})$ in , . Let $f_P(M) = (ZW, \mu, \varphi_P)$ and t be the tuple on ZW with $t[A] = 1$ for each $A \in ZW$. It is easily seen that t is in $\varphi_P(2)$. Indeed, by the construction of M, for each tuple t' in $\varphi(2)$, there exists $P' = P_X(Y^{persis})$ in , with $Z \not\subseteq X$ such that $t'[A] = 1$ if $A \in X$ and $t'[A] = 0$ otherwise. Since $t[A] = 1$ for each $A \in Z$, it is now clear by such a construction that $t'[Z] \neq t[Z]$ for each $t' \in \varphi(2)$. Hence, by persistence, t is derived from the tuple $(1, \ldots, 1)$ in $\varphi(1)$ by the persistence $P = P_Z(W^{persis})$, i.e., t is in $\varphi_P(2)$.

We now prove that $\overline{M}$ is not a closure of M under $P_Z(W^{persis})$ by showing that $t \notin \pi_{ZW}\overline{\varphi}(2)$. In order to do this, since $t \notin \pi_{ZW}(\varphi(2))$, by (*), we need only to show that for each $P' = P_X(Y^{persis}) \in$, , there does not exist $t' \in \overline{\varphi}(2)$ such that $t'[XY] \in \varphi_{P'}(2)$ and $t'[ZW] = t$. Let $P' = P_X(Y^{persis})$ be in , . Two cases arise: (a) $Z \not\subseteq X$ and (b) $Z \subseteq X$ but $W \not\subseteq (X \setminus Z) \cup Y$. Consider (a). Note that by the construction of $\varphi(2)$ and the persistence assumption, $\varphi_{P'}(2) = \pi_{XY}(\varphi(2))$, i.e., no tuple is derived from the tuple $(1, \ldots, 1)$ in $\varphi(1)$, and hence, we know that $t \notin \pi_{ZW}(\varphi(2))$. Consider (b). In this case, there exists an attribute $A \in W$ such that $A \in (R \setminus XY)$. Let t' be in $\varphi_{P'}(2)$. This tuple is either already in $\varphi(2)$ (in this case, $t'[ZW] \neq t$ as we reasoned above), or it is derived from the tuple $(1, \ldots, 1)$ in $\varphi(1)$ by P'. In this latter case, we know that $t'[XY] = (1, \ldots, 1)$ and $t'[R \setminus XY] = (0, \ldots, 0)$. Since $t[ZW] = (1, \ldots, 1)$ and there exists $A \in W$ such that $A \in (R \setminus XY)$, it follows that $t'[ZW] \neq t$. □

A.4 Proof of Theorem 2

Proof. Let Q be a correctly-typed query on a database DB with a collection , of point-based assumptions expressed in $flat-MQL^F$, and Q' the query obtained by Algorithm 1 with Q and , as inputs. In order to prove the theorem, we show that $Q'[DB] = Q[DB, ,]$. Suppose $DB = \{M_1, \ldots, M_k\}$. We know that, by definition, $Q[DB, ,] = \overline{Q}[\overline{DB}] = \overline{Q}[\overline{M_1}, \ldots, \overline{M_k}]$, where each $\overline{M_i}$ is the minimal closure of M_i. The query Q, as any query in MQL^F, has the form $\{x_1, \ldots, x_n, t : \nu \mid \Psi(x_1, \ldots, x_n, t)\}$. However, for simplicity of the proof, we generalize the query to have more than one free temporal variable. Hence, Q has the general form: $\{x_1, \ldots, x_n, t_1 : \nu_1, \ldots, t_m : \nu_m \mid \Psi(x_1, \ldots, x_n, t_1, \ldots, t_m)\}$. We use induction on the structure of the formula Ψ. Since the transformation is only concerned with atomic formulas of the type $M()$, the case when $\Psi = M_1(x_1, \ldots, x_n, t)$ where M_1 is an arbitrary module in DB is the crucial step of the proof. In this case, to evaluate Q on the closure of DB, i.e. the closure of M_1, it means considering $\overline{M_1}()$ instead of $M_1()$ as a predicate in Q. Hence, $\overline{Q}[\overline{DB}] = \{x_1, \ldots, x_n, t \mid \overline{M_1}(x_1, \ldots, x_n, t)\} = \overline{M_1}$. From the algorithm we have: $Q' = \{x_1, \ldots, x_n, t \mid \Psi^{\mathtt{M}}_{P_1}(x_1, \ldots, x_n, t) \vee \ldots \vee \Psi^{\mathtt{M}}_{P_l}(x_1, \ldots, x_n, t)\}$ where P_1, ..., P_l are all the assumptions in , containing all the attributes in $\mathtt{M}$. We now show that $\overline{Q}[\overline{DB}] = Q'[DB]$ in this case. By definition, $\overline{M_1}$ is the minimal module having the same temporal module scheme as M_1 such that $f_{P_i}(\mathtt{M}_1) \preceq \overline{M_1}$ for each $1 \leq i \leq l$. We know that the assumption mappings can be expressed in $flat-MQL^F$; for example,

$f_{P_1}(\mathtt{M}_1) = \{x_1, \ldots, x_n, t \mid \Psi^{\mathtt{M}_1}_{P_1}(x_1, \ldots, x_n, t)\}$. By Proposition 1, $\overline{M_1}$ is the temporal module whose windowing function is given by $\varphi_{\mathtt{M}_1}(t) = \{x_1, \ldots, x_n \mid \Psi^{\mathtt{M}}_{P_1}(x_1, \ldots, x_n, t) \vee \ldots \vee \Psi^{\mathtt{M}}_{P_l}(x_1, \ldots, x_n, t)\}$ for each time t, since the logical or ($\vee$) corresponds to the union operation. From this fact it is clear that the evaluation of Q' on DB gives exactly the same tuples present in $\overline{M_1}$ and this concludes the proof of the case when the atomic formula is $M_1()$. Other cases of atomic formulas regard comparisons among variables: $x_i < x_j$, $x_i = x_j$, $t_i < t_j$, $t_i = t_j$, and $\mathbf{IntSec}_{\mu,\nu}(t_i, t_j)$. When Ψ is one of these formulas, no transformation is done and the answer is independent from DB. Hence, the theorem holds in these cases. Given that the theorem holds when Ψ is an atomic formula, we can easily apply the induction step to complete the induction. However, the induction is trivial since the transformation does not affect other constructs of a given formula. We thus omit the induction step and conclude that the theorem holds. □

A.5 Proof of Proposition 5

Proof. To prove the proposition, we first argue that it is decidable if a type μ can be reached from type ν via the interval-based assumption $I = I_X(Y^{conv})$, where the semantics of *conv* is specified by a MQL^F formula template. Indeed, by condition 3 in the definition of the semantics for conversion methods, a type μ can be reached from type ν via I if and only if it can be reached via $I' = I_X(A^{conv})$, where A is a single attribute in Y. Furthermore, by condition 4 in the same definition, the type μ can be reached from type ν via $I_X(A^{conv})$ if and only if there exists a temporal module $M = (XA, \mu, \varphi)$ with the condition that $t[X]$ is a fixed constant for all tuples t in the module, and such that $\{x_1, \ldots, x_k, t{:}\nu \mid \Psi^{\mathtt{M},\nu}_{I_X(A^{conv})}\}$ is not empty. For each predicate $\mathtt{M}(x_1, \ldots, x_n, t)$ appearing in $\Psi^{\mathtt{M},\nu}_{I_X(A^{conv})}$, we may simply replace it with $M_{x_n}(t) \wedge x_1 = a_1 \wedge \cdots \wedge x_{n-1} = a_{n-1}$ assuming that the constant for X is $a_1, \ldots, a_{n-1}$. (Here, without loss of generality, we assume that A is the last attribute in $\mathtt{M}$.) Furthermore, replace any quantifier $\exists x$ and $\forall x$ by $\exists M_x$ and $\forall M_x$, and replace $x = y$ by $\forall t'(M_x(t') \equiv M_y(t'))$. Note that $t' < t''$, where t' and t'' are time variables, will be unchanged in the formula. Call such changed formula Ψ', which is in monadic second-order logic. It is easily seen that the formula Ψ' is valid iff the original Ψ is valid. It is known [GHR94, page 558] that the validity of the formulas in this logic is decidable since the linear order we use is a discrete (integer) order. Hence, it is decidable if $(\Psi^{\mathtt{M},\nu}_{I_X(A^{conv})})$ can be satisfied by a temporal module, i.e., whether the query $\{x_1, \ldots, x_k, t{:}\nu \mid \Psi^{\mathtt{M},\nu}_{I_X(A^{conv})}\}$ is always empty or not. Thus we conclude that, for a given arbitrary assumption I of the form $I_X(Y^{conv})$ and types μ and ν, it is decidable if μ can be reached from ν via I.

It is now easily seen that, for a given arbitrary assumption I and types μ and ν, it is decidable if μ can be reached from ν via I. [The reason is that the semantics of an interval-based assumption that involve more than one conversion method, is the join of the semantics of each conversion method.] From this, we conclude that the problem of whether a query is doable is decidable.

The lower bound of the complexity of the decision procedure for the above monadic second-order logic formula is $2^{2^{cn}}$ according to [FR79, page 3]. □

A.6 Proof of Theorem 3

Proof. Let Q be a query on a database DB with a collection , of interval-based assumptions expressed in MQL^F, and Q' the query obtained by Algorithm 2 with Q and , as inputs. In order to prove the theorem, we show that $Q'[DB] = Q[DB, ,]$. Suppose $DB = \{M_1, \ldots, M_k\}$. Let $\nu_1, \ldots, \nu_r$ be all the temporal types appearing in the database DB and assumptions in , and in the query Q. Let

$\overline{DB} = \{M_{1,\nu_1}, \ldots, M_{1,\nu_r}, \ldots, M_{k,\nu_1}, \ldots, M_{k,\nu_r}\}$, where each M_{i,ν_j} is the union of the modules $f_I(M_i, \nu_j)$ for each interval-based assumption $I \in$, involving all the attributes in M_i. Let $\overline{Q}$ be the query obtained by changing each $\mathtt{M}_i(x_1, \ldots, x_k, t)$ to $\mathtt{M}_{i,\nu}(x_1, \ldots, x_k, t)$ in Q, where ν is the type of t. By definition, $Q[DB, ,] = \overline{Q}[\overline{DB}]$. We only need to show that $Q'[DB] = \overline{Q}[\overline{DB}]$.

The query Q, as any query in MQL^F, has the form $\{x_1, \ldots, x_n, t:\nu \mid \Psi(x_1, \ldots, x_n, t)\}$. However, for simplicity of the proof, we generalize the query to have more than one free temporal variable. Hence, Q has the general form: $\{x_1, \ldots, x_n, t_1 : \nu_1, \ldots, t_m : \nu_m \mid \Psi(x_1, \ldots, x_n, t_1, \ldots, t_m)\}$. We use induction on the structure of the formula Ψ. The induction proof differs from that of Theorem 2 only for the case when $\Psi = M(x_1, \ldots, x_k, t:\nu)$ where M is an arbitrary module in DB. In this case we have $\overline{Q} = \{x_1, \ldots, x_k, t:\nu \mid M_\nu(x_1, \ldots, x_k, t)\}$ that, evaluated on $\overline{DB}$ gives $\overline{Q}[\overline{DB}] = M_\nu$. From the algorithm we have: $Q' = \{x_1, \ldots, x_k, t:\nu \mid \Psi_{I_1}^{\mathtt{M},\nu}(x_1, \ldots, x_k, t) \vee \ldots \vee \Psi_{I_l}^{\mathtt{M},\nu}(x_1, \ldots, x_k, t)\}$ where I_1, $\ldots$, I_l are the interval-based assumptions in , containing all the attributes in $\mathtt{M}$. We now show that $\overline{Q}[\overline{DB}] = Q'[DB]$ in this case. By definition, M_ν is the module having the same attributes and temporal type as $\mathtt{M}$ and defined as $f_{I_1}(\mathtt{M}, \nu) \cup^T \ldots \cup^T f_{I_l}(\mathtt{M}, \nu)$. We know that the assumption mappings can be expressed in MQL^F; for example, $f_{I_1}(\mathtt{M}, \nu) = \{x_1, \ldots, x_k, t:\nu \mid \Psi_{I_1}^{\mathtt{M},\nu}(x_1, \ldots, x_k, t)\}$. Hence, M_ν is such that its windowing function $\varphi(t) = \{x_1, \ldots, x_k \mid \Psi_{I_1}^{\mathtt{M},\nu}(x_1, \ldots, x_k, t) \vee \ldots \vee \Psi_{I_l}^{\mathtt{M},\nu}(x_1, \ldots, x_k, t)\}$ for each time t. From this fact it is clear that the evaluation of Q' on DB gives exactly the same tuples present in M_ν and this concludes the proof of this case. The rest of the induction proof is trivial as in the proof of Theorem 2 and thus omitted.

□

A.7 Proof of Proposition 6

Proof. If the query Q is on a generic database $M_1, \ldots, M_k$ the only predicates appearing in Q will be $M_i()$ for any $1 \leq i \leq k$. Each predicate instance corresponds to the closure of a module in the database accordingly with the point-based assumptions or it corresponds to a module derived accordingly to interval(or interval + point)-based assumptions. Since the assumptions are assumed to be safe, the values in the minimal closures of these modules will be either from the database or from function application on values of the database. Hence, the syntactic conditions of safety of Q and , ensure that the values resulting from the query are either from the database or from a bounded number of function applications on values in the database, and we can conclude that Q is embedded domain independent wrt , . □

A.8 Proof of Theorem 5

Proof. Given a general query Q on a set of modules $\mathtt{M}_1, \ldots, \mathtt{M}_k$ with assumptions , , by Theorem 4 we know that its answer is the same as that of the query Q' on $\mathtt{M}_1, \ldots, \mathtt{M}_k$ without assumptions, where Q' is obtained by the transformations of Algorithms 1 and 2 using assumptions , . Hence, without loss of generality, we will only show that for each query on a database (without assumptions), its answer is 1st-order finitely partitioned.

For each 1fp-module, since the number of tuples in a temporal module are assumed finite (i.e., the set $\bigcup_{i \geq 1} \varphi(i)$ is finite), $\mathtt{M}$ can be represented by the following formula

$$\psi_{\mathtt{M}}(w_1, \ldots, w_n) = (w_1 = a_{11} \wedge \ldots \wedge w_n = a_{1n} \wedge \chi_1(t)) \vee \ldots \vee (w_1 = a_{r1} \wedge \ldots \wedge w_n = a_{rn} \wedge \chi_r(t)),$$

where each χ_j is a MQL^F formula involving only temporal variables and temporal predicates (**IntSec** and $<$). Each subformula in the disjunction identifies an n-tuple, and the formula $\chi_j(t)$ identifies the

set of ticks at which that tuple is valid. For example, the subformula $(w_1 = a \ \wedge \ w_2 = b \ \wedge \ ((1 < t \ \wedge \ t < 4) \ \vee \ t > 8))$ represents the tuple (a, b) in $\mathtt{M}$ as present at each tick between 1 and 4, and at each tick greater than 8. Note that the number r of subformulas in the above disjunction equals the number of different tuples in a module. Hence, $\Psi_{\mathtt{M}}$ has a finite length for each 1fp-module $\mathtt{M}$.
Let $Q' = \{x_1 \ldots x_m, t \mid \psi(x_1, \ldots, x_m, t)\}$. Let $\psi'(x_1, \ldots, x_m, t)$ be the formula obtained from $\psi()$ by substituting each predicate $\mathtt{M}(w_1, \ldots, w_n, t')$ appearing in it by the corresponding formula $\psi_{\mathtt{M}}(w_1, \ldots, w_n, t')$. All the terms in ψ' are of one of the forms $x_i \sim x_j$, where $\sim \in \{=, \neq, +, *, -, /\}$ and $t_l \sim t_m$ where $\sim \in \{=, \neq, <, >\}$. It is always possible to transform ψ' into a formula ψ'' having the following structure:

$$\psi''(x_1, \ldots, x_k, t) = (\psi''_1(x_1, \ldots, x_k) \ \wedge \ \chi_1(t)) \ \vee \ \ldots \vee (\psi''_s(x_1, \ldots, x_k) \ \wedge \ \chi_s(t)).$$

Each $\psi''_j(x_1, \ldots, x_k)$ is a first-order formula on the data domain with free variables $x_1, \ldots, x_k$ and involving only terms $x_i \sim x_j$. It intuitively identifies a set of tuples in the answer. Each $\chi_j(t)$ is a first-order formula on the temporal domain with the only free variable t and involving only terms $t_l \sim t_m$. It intuitively identifies the set of ticks at which the tuples specified in ψ''_j are in the answer. This representation of the answer satisfies the definition of 1st-order finitely partitioned module. Indeed, there are a finite number of sets of ticks, each identified by a 1st-order formula $\chi'_j(t)$, forming a partition of the target type and such that the value of the windowing function is the same for each set. □

A.9 Proof of Theorem 6

Proof. From the proof of Theorem 5, the answer of a MQL^F query takes the form:

$$\psi''(x_1, \ldots, x_k, t) = (\psi''_1(x_1, \ldots, x_k) \ \wedge \ \chi_1(t)) \ \vee \ \ldots \vee (\psi''_s(x_1, \ldots, x_k) \ \wedge \ \chi_s(t)).$$

Since now each $\chi_i(t)$ only involves one type of temporal variables and no **IntSec** predicate is used, we may apply quantifier elimination procedures [Coo72] to reduce each χ_i to χ'_i: a conjunction of temporal terms with t as the only free variable. χ'_i intuitively identifies a finite set of intervals on time at which the tuples must be in the answer module. For example, if $\psi''_j = (\exists y > 0 \ x_1 + y = x_2 \ \wedge \ x_2 = 10)$ and $\chi'_j(t) = t > 8$, the answer module will contain at each tick after 8, all the possible tuples having 10 as value of the second attribute and a value less then 10 for the first attribute. Take t_{max} as the greatest temporal constant appearing in all χ'_i. For each tick j after t_{max}, the formula ψ'' identifies the same set of tuples as for tick t_{max}. Hence, the answer is eventually uniform, and this concludes the proof. □