Jan Chomicki, Gunter Saake,
Ron van der Meyden (Eds.)

Logics for
Emerging Applications of Databases

Springer

Contents

List of Contributors

David Toman
School of Computer Science
University Waterloo
200 University Avenue West
Waterloo, ON N2L 3G1
Canada
david@uwaterloo.ca

Logical Data Expiration

David Toman

School of Computer Science, University Waterloo
Waterloo, ON N2L 3G1, Canada
email: david@uwaterloo.ca

Abstract. Data expiration is an essential component of data warehousing solutions: whenever large amounts of data are repeatedly collected over a period of time, it is essential to have a clear approach to identifying parts of the data no-longer needed and a policy that allows disposing and/or archiving these parts of the data. Such policies are necessary even if adding storage to accommodate an ever-growing collection of data were possible, since the growing amount of data needs to be examined during querying and in turn leads to deterioration of query performance over time.

The approaches to data expiration range from ad-hoc administrative policies or regulations to sophisticated data analysis-based techniques. The approaches have, however, one thing in common: intuitively, they try to identify the parts of the data collection that are not needed in the future. The key to deciding if a piece of information will be needed in the future lies in identifying what queries can be asked over the collection of data and how the collection can evolve from its current state. The various techniques proposed in the literature differ in the way they identify parts of data no longer needed.

This chapter formalizes the notion of data expiration in terms of how the data is used to answer queries. We survey existing approaches to the problem in a unified framework and discuss their features and limits, and the limits of data expiration based techniques in general. The particular focus of the chapter is on comparing the *space performance* of various data expiration methods.

1 Introduction

Modification and evolution of data over time is essential to most applications of information systems, databases, and data warehouses. Change of data is, however, not restricted to information systems and databases: state transitions of programs can also be viewed as updates of internal data structure(s). In general, programs

- query the data they manage using a *query language*; this language ranges from primitive look-ups of memory locations fully under control of an application program to sophisticated declarative queries over large data warehouses;
- modify the data; again the changes range from writing a single value to a memory location to refreshing data in a data warehouse.

It is sufficient for most applications to keep track of the *current state* of the data only. However, there are many important cases where access to data that have existed in some *past state* of the system is necessary. Typical situations in which access to the historical data is required are:

- monitoring applications
- data warehouse evolution analysis
- enforcement of dynamic/temporal integrity constraints

The access to historical data can be conceptually formalized by considering queries to be expressed in a *query language* over the history of the evolution of the system's state: a sequence of consecutive snapshots of the data representing this state. This model provides an easy to understand conceptual foundation for accessing and querying the history. However, if such a history was stored naively, e.g., by making a copy of the current state before every modification, storage and subsequent retrieval costs would quickly became prohibitive with the progression of time. Application programs must therefore attack such an unbounded growth. There are two general approaches:

1. The application assumes all responsibility for all aspects of data in particular for discarding data no longer needed.
2. The application delegates the responsibility of data removal to a *data expiration* (or garbage collection [27]) subsystem that, based on the knowledge of future queries and updates the application may perform, attempts to analyze the stored data and discard any parts no longer needed.

The second approach is becoming much more popular; its applications range from data warehousing solutions designed to maintain historical data [49,50] to compilers and execution environments of modern programming languages [27]. It embodies the usual advantages of using a higher-level interface (e.g., a database system) to manage data.

In this chapter we study techniques that can be used to limit the growth of histories. In general, the techniques attempt to *encode* the history in some compact way while preserving answers to queries issued by an application. Note that the encoding must carry sufficient information not only to be able to provide answers to an application query at a particular point in time, but also to retain this ability in all possible extensions of the history.

While data expiration is the main focus of the chapter, the results presented here are applicable in other fields. In particular, the goals of recent research in the area of *streaming data management systems* and on *queries over continuous data streams* [7] are very similar to those of data expiration: the continuous query processing techniques attempt to limit the amount of data needed to answer a given query over an ever-growing (append-only) data stream. Similar to data expiration, the approaches try to be selective in remembering only the necessary parts of the data stream (called *synopses*) in a limited space. For example, recent results on streaming queries [5] parallel

results on enforcement of temporal integrity constraints [10,12], a special case of data expiration (cf. Section 4.1 in this chapter).

The chapter is organized as follows. Section 2 defines a general framework for data expiration. Section 3 reviews policy-based approaches to data expiration. Section 4 outlines two principal approaches to query-driven expiration. Section 5 discusses the implications of allowing queries over potentially infinite histories. The chapter concludes with open problems and directions for future research.

While the chapter attempts to be self-contained and provides most of the necessary definitions, it still assumes a certain level of familiarity with temporal query languages (for a survey see [13]), with basic results on temporal logic [16], and with the relational model [2].

2   Framework for Data Expiration

In this section we introduce a basic framework in which data expiration can be studied. The framework is centered around two main notions, the notion of a *history* of evolution of an system, and the notion of an *expiration operator*.

2.1   Histories

We formalize the evolution of a system in terms of *histories*: Let S_i be the description of the system state at time i (in some data model). With the progression of time the system makes transitions from state S_{i-1} to state S_i. This transition is usually reflected in modification of the data associated with the state S_{i-1}. We (conceptually) record the sequence of such transitions as a *history* of the system, a finite[1] sequence

$$H = \langle S_0, S_1, \ldots, S_m \rangle$$

of system states. We use $\langle\ \rangle$ to denote the empty history and $H; S$ to denote the extension of H by an additional state S. For example, we can record

- changes of variable values in a program: each time a value is updated in a variable, a new state of the system is created;
- enrollments of students in classes: each term produces a new state of our system that records who is taking what class in the particular term.

The history records *discrete* changes of a system as a linearly ordered sequence of states S_i. The subscripts i are therefore drawn from a *temporal domain*, a discrete, linearly ordered structure $(T, <)$, and are called the *time instants*[2]. Intuitively, the history records the facts that the system was in

---

[1] We consider the implications of relaxing this condition in Section 5.
[2] These values may be different from the actual *real, wall-clock time* values at which the state change occurred.

state S_i at time i. The discrete nature of of the temporal domain also yields the notion of a previous or next state of the system in terms of consecutive time instants. Therefore, in the remainder of this chapter we assume, without loss of generality, that time is modeled by natural numbers with the usual linear ordering (and thus we use natural numbers to index the states in histories). From a temporal database point of view, the histories can be viewed as append-only, transaction-time temporal databases.

The histories allow us to formulate questions that span multiple system states. For example, in a history generated by variable assignments, we can ask the following questions:

- Has variable x been modified? Has it been modified before y?
- What were the values of x so far? What were the values of x when the value of y was 1 (e.g., when the system was executing a critical section)?
- Were the values of x increasing? If not, what were the cases of decrease?

Similarly, for the history of class enrollments we can ask

- Who was taking classes in the first term (first ever—i.e., since the opening of the school)? In the last 5 terms?
- Which students have taken the same class twice?

The queries over the histories are usually formulated in a *temporalized* version of a (query) language used by the application to interrogate individual states. The temporal queries rely on the time instants attached to the individual states of the history to gain access to past states of the history and/or to compare the *relative age* of various pieces of the data represented in the history. The queries must, however, respect the structure of time: the only defined relation on this structure is the discrete linear order ($<$). Thus all relationships between time instants must be defined in terms of this order or must depend on values *stored* in the states S_i. This is, in particular, true for the *real-time wall clock* should we need one (cf. Section 4.1). This also means that queries must not distinguish between discrete linearly ordered sets; hence our choice of natural numbers for time instants.

2.2   Expiration Operators

In practice, we often cannot afford to *store* the whole history explicitly. Therefore, various history compression and/or *data expiration methods* have been devised. These approaches can be characterized in terms of an *expiration operator*. More precisely, given a temporal query language $\mathcal{L}$ over histories of system states and a query $Q \in \mathcal{L}$ we define an *expiration operator* as a mapping

$$\mathcal{E} : Q \to (0^{\mathcal{E}}, \Delta^{\mathcal{E}}, Q^{\mathcal{E}}).$$

Given a query Q, the mapping produces three components. The first two components, $0^{\mathcal{E}}$ and $\Delta^{\mathcal{E}}$, provide the actual inductive definition of the expiration

operator. These two functions define the result of the expiration process with respect to extensions of a history:

$$\mathcal{E}(\langle\,\rangle) = 0^{\mathcal{E}} \qquad \text{(initial state)}$$
$$\mathcal{E}(H;S) = \Delta^{\mathcal{E}}(\mathcal{E}(H), S) \qquad \text{(extension maintenance)}$$

We call the result $\mathcal{E}(H)$ the *residual history*.

The third component, $Q^{\mathcal{E}}$, is a query over the residual history that mimics the original query Q. The three components must maintain the following soundness condition:

$$Q(H) = Q^{\mathcal{E}}(\mathcal{E}(H)) \qquad \text{(answer preservation)}$$

In general, we do not restrict the structure of $\mathcal{E}(H)$; in particular we do not insist that this value has to conform to a particular data model, e.g., the data model of H.

The following examples show that our formalization is general enough to capture the two extremes: keeping the *whole history* and keeping the *current state* only:

Example 1 (Retaining Complete History). One extreme in defining the expiration operator $\mathcal{E}_{\mathrm{id}}$ is defining

$$0^{\mathcal{E}_{\mathrm{id}}} = \langle\,\rangle$$
$$\Delta^{\mathcal{E}_{\mathrm{id}}} = \lambda H \lambda S.H;S \qquad \text{where} \qquad Q^{\mathcal{E}_{\mathrm{id}}} = Q$$

This trivial operator simply keeps the whole history intact and maps queries to themselves. It is easy to verify that this definition trivially satisfies the requirements placed on expiration operators above.

Example 2 (Retaining Current State Only). The other extreme is keeping only the current (last) state of the history. Here, the query language L may only contain queries that reference the current state; otherwise, there is no possibility to satisfy the *answer preservation property*. The definition of the operator $\mathcal{E}_{\mathrm{now}}$ is then as follows:

$$0^{\mathcal{E}_{\mathrm{now}}} = \langle\,\rangle$$
$$\Delta^{\mathcal{E}_{\mathrm{now}}} = \lambda H \lambda S.\langle S \rangle \qquad \text{where} \qquad Q^{\mathcal{E}_{\mathrm{now}}} = Q$$

Another common approach to storing histories is the use of *compression*:

Example 3 (Compression). Let compress and decompress be two functions implementing lossless compression of histories. Then the triple

$$0^{\mathcal{E}_{\mathrm{compress}}} = \mathrm{compress}(\langle\,\rangle)$$
$$\Delta^{\mathcal{E}_{\mathrm{compress}}} = \lambda H \lambda S.\mathrm{compress}(\mathrm{decompress}(H);S)$$
$$Q^{\mathcal{E}_{\mathrm{compress}}} = \lambda H.Q(\mathrm{decompress}(H))$$

defines an expiration operator.

The three examples above have shown operators that do not depend on the query Q. However, it is easy to see that knowing queries that are asked over the history in advance can (and will) improve the chances of an expiration operator to remove unneeded data from the history.

Example 4. Let H be a history of relational databases with the schema containing two (unary) predicate symbols $R(x)$ and $S(x)$. Then, given a (first order) query[3] $\{(t, x) : R(t, x)\}$, it is easy to see that an expiration operator can be defined by mapping the original history

$$H = \langle (\mathbf{R}, \mathbf{S})_0, (\mathbf{R}, \mathbf{S})_1, \ldots, (\mathbf{R}, \mathbf{S})_n \rangle$$

to the history

$$\mathcal{E}(H) = \langle \mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_n \rangle,$$

where $\mathbf{R}_i$ and $\mathbf{S}_i$ are instances of the predicate symbols R and S, at time i, respectively. This mapping satisfies all the requirements imposed on expiration operators.

In general, we can use a much finer classification of data objects, depending on what query or queries are asked over the original history. Other typical examples include classifying predicates to *historical* (for which the whole history is to be kept) and *current* (for which only the last state is kept), etc. Note that if the application doesn't ask queries over the history, we don't have to keep any (historical) data at all.

2.3   Relational Setting

In the rest of this chapter we constrain our attention to the *relational setting*; we assume that the system states S_i can be described by a finite relational database D_i over a *fixed* schema ρ.

This restriction does not limit the applicability of techniques and results presented in this chapter to other data models. Indeed, in many cases, other models of finite data collections can be naturally embedded into the relational model (e.g., an object-oriented database can be viewed as a relational database with unary and binary relations).

In this setting a history can be defined as follows:

Definition 1 (History). Let ρ be a relational signature. A *history* H is a finite integer-indexed sequence of databases

$$H = \langle D_0, D_1, \ldots, D_n \rangle$$

where D_i is a standard relational database over ρ. We call D_i a *state* of H at a *time instant* i.

The *data domain* $\mathbf{D}_H$ of a history H is the union of all (data) values that appear in any relation in D_i at any time instant; the *temporal domain* $\mathbf{T}_H$ is the set of all time instants that appear as indices in the history H.

---

[3] Section 4.2 gives a precise definition of semantics for such queries.

Note that the sets $\mathbf{T}_H$ and $\mathbf{D}_H$ represent the *active domains* for the temporal and data sorts, respectively. In particular, given a (linearly ordered) structure $(T, <)$ that models time we have $\mathbf{T}_H \subseteq T$.

Example 5. Let $\mathrm{TA}(x, y)$ be a binary relation recording student names and classes for which they worked as Teaching Assistants. An example history (indexed, e.g., by the academic term) may look as follows:

$$\left\{(\mathrm{John}, \mathrm{CS448})\right\} \quad \left\{\begin{matrix}(\mathrm{John}, \mathrm{CS448})\\(\mathrm{Sue} \ , \mathrm{CS234})\end{matrix}\right\} \quad \left\{(\mathrm{John}, \mathrm{CS448})\right\}$$

$$0 \qquad\qquad 1 \qquad\qquad 2$$

A history H can be *extended* by adding a database D to the end of the sequence. The database D then becomes a new state D_j of H and we require that the time instant j associated with D is strictly larger than the last time instant associated with any state of H. This way, the new state D effectively becomes the new current state of H. This process can be repeated arbitrarily many times.

Example 6. The history in Example 5 can be extended by a database representing an additional term as follows:

$$\left\{(\mathrm{John}, \mathrm{CS448})\right\} \quad \left\{\begin{matrix}(\mathrm{John}, \mathrm{CS448})\\(\mathrm{Sue} \ , \mathrm{CS234})\end{matrix}\right\} \quad \left\{(\mathrm{John}, \mathrm{CS448})\right\} \quad \left\{(\mathrm{Sue}, \mathrm{CS234})\right\}$$

$$0 \qquad\qquad 1 \qquad\qquad 2 \qquad\qquad 3$$

This history is used for the remaining examples in the chapter.

In general, we can inductively define an extension of a given history by another history—a *concatenation of histories*. Let H' be the sequence of states successively added to H up to a given time instant. We call H' a *suffix* of H and write $H; H'$ for the extension of H by H'.

Query Languages. The choice of the relational model for describing the states of a system allows us to use a declarative language—the *relational calculus* (first-order logic FOL)—to query the individual states of the system. There are two standard extensions of FOL to histories (and, in general, temporal structures):

1. by adding temporal operators, or
2. by adding explicit references to time instants.

The first choice leads to various *first-order temporal logics* while the second approach yields a *two-sorted first-order logic* (temporal relational calculus). Both these options have been extensively investigated; for a survey see [13].

Finite Histories vs. Infinite Extensions of Histories. When considering the semantics of a query language over histories, we are faced with an additional choice:

1. We can assume that the current history contains *complete information* about the system and define semantics of queries with respect to the current (finite) history. This approach is similar to the closed-world assumption (CWA) and is equivalent to choosing the *active domain* semantics for queries [2].
2. Alternatively, we can treat the current history as a finite prefix of an infinite (often called *complete*) history. In this setting the semantics of queries is defined with respect to infinite extensions (completions) of the current history and is similar to the open-world assumption[4] (OWA).

In most parts of this chapter we restrict our attention to the *active domain* semantics only. This restriction postulates that, for the purpose of query answering, the only data values and time instants that exist are those present in the history. Note that, in particular, all the quantifiers in the definitions of the temporal operators range *over the active temporal domain*: the set of time instants present in the finite history H. All the expiration techniques are developed with this restriction in mind. In Section 5 we discuss the implications of relaxing the restriction.

Real Time vs. Sequence Numbering. The other assumption we use is that the structure of the temporal domain is based on the discrete ordering of natural numbers (apart from the discussion of *Metric TL* in Section 4.1).

2.4   Properties of Expiration Operators

The methods differ in the query language they are able to handle and in the space needed to store the residual database, $|\mathcal{E}(H)|$. This last feature can be measured with respect to several parameters:

- the size of the history itself, $|H|$,
- the size of the active data domain, $|\mathbf{D}_H|$,
- the length of the history, $|\mathbf{T}_H|$, and
- the size of the query, $|Q|$.

An expiration method provides a *bounded encoding of a history* if $|\mathcal{E}(H)|$ does not depend on the length of the history (more precisely, $|\mathcal{E}(H)|$ is bounded by a function constant with respect to $|\mathbf{T}_H|$) [10]. The expiration operators $\mathcal{E}^{\mathrm{id}}$ and $\mathcal{E}^{\mathrm{compress}}$ introduced in Section 2.2 are not bounded while the operator

---

[4] The open-world assumption, however, is only assumed for the *extensions* of the history; the individual databases that describe states in the current history are still considered to contain complete information about those states.

$\mathcal{E}^{\text{now}}$ is bounded. Note however, the last operator works only for a very limited set of queries. It is easy to see that expiration operators utilizing *lossless encoding* of histories cannot be bounded. This is in particular true for the interval encoding of temporal databases (and thus histories) [42].

The ability to expire data from a history depends on the expressive power of the query language in which queries over the history are formulated. In particular, allowing an arbitrary number of *ad-hoc* queries precludes any possibility of expiring data:

Proposition 1. *Finite relational structures can be completely characterized by first-order queries.*

Thus, for common temporal query languages, the observation leaves us with two essential options:

1. we can adopt an *administrative* solution and expire a given history using a set of *policies* independent of queries. Now we can still allow ad-hoc querying of the history. However, we would like to fail queries that try to access the already expired values, perhaps informing the application that the returned answer may be only approximate, or
2. we can adopt a query driven data expiration technique. Such a technique, however, can only work for a fixed set of queries *known in advance.*

Alternatively, we can restrict the expressiveness of the temporal query language in such a way that it is no longer able to distinguish sufficiently large finite histories. However, this option usually leads to restrictions on *nesting of quantifiers* in the formula and in essence is equivalent to restricting ourselves to a finite set of queries.

3   Administrative Approaches to Data Expiration

One approach to expiring data from histories, and, in turn, to defining *expiration operators*, can be based on *query-independent expiration policies.* However, when data is removed from a history in a query-independent way, the system should be able to characterize queries whose answers are not affected.

3.1   Expiration Operators vs. Materialized Views

A query-independent expiration operator can be thought of as a fixed view $\mathcal{E}(H)$ defined over the original history H. Thus, we can transfer results and techniques developed for maintaining materialized views and for answering queries using these views to the problem of administrative data expiration.

Answering Queries using Views. The ability to answer an (ad-hoc) query Q over the residual history $\mathcal{E}(H)$ reduces to deciding whether there is an equivalent query Q' that can be expressed solely using the view $\mathcal{E}(H)$:

$$Q(H) = Q'(\mathcal{E}(H)) \ \text{ for all histories } H.$$

The problem of *answering queries using views* [29] has been studied extensively; for a recent survey see [21]. Application of these techniques to expiration of database histories, however, requires reasoning about queries and views whose definitions take into account the inherent linearly ordered structure of time. The problem of answering queries over views *with ordered data domains* has been recently addressed by Afrati, et al, [3].

Maintenance of Materialized Views. The second essential requirement imposed on the view of the history defined by an expiration operator is that we must be able to maintain the view(s) that define the residual history under arbitrary history extensions *without access to the original history*.

Similarly to the problem of answering queries using views, the issues connected with *maintenance of materialized views* have been widely studied [20]. Standard techniques for maintenance of materialized views use sets of tuples inserted and deleted to/from base relations (so called *delta relations*) to incrementally update the instance of the materialized view.

These techniques, however, often need access to the whole underlying database to perform the update. This is contrary to the goals of expiration operators: the goal of data expiration is to discard the original history (source database) and maintain only the result of the operator (a materialized view). This requirement restricts the allowed views of the original history to those that are *self-maintainable* [37]: the update of the view can be computed from the current instance of the view and the delta relations.

3.2   Vacuuming Database Histories

Probably the most common approach to defining an expiration operator independently of the application queries is the *history truncation* or *cutoff point* approaches. In the world of *materialized views*, these approaches correspond to *temporal selection*, single-table views defined over the schema of the history. Such selection-based views define in two major variants of cutoff-based expiration policies:

1. policies based on a fixed, *absolute cutoff point*, or
2. policies based on a *now-relative cutoff point*.

It is easy to see, that the first choice is a generalization of the $\mathcal{E}^{\mathrm{id}}$ operator and does not lead to a bounded history encoding. The second choice—a sliding window style generalization of the $\mathcal{E}^{\mathrm{now}}$ operator—keeps a consecutive but

fixed number of past states. In this case the length of the remaining history is independent of the history length.

The interaction of these extremely simple policies for data expiration with queries can be summarized as follows:

Proposition 2. *It is undecidable whether a first-order query refers to fixed past only.*

On the other hand, we can *detect* at query evaluation time whether a query tried to access *expired parts* of the history and issue an error message.

Jensen, et al, [26,41] extended the allowed specifications of expiration policies to include expressions of the form

- $\rho(R) : e$ (a *remove* specification), and
- $\kappa(R) : e$ (a *keep* specification).

where R is a relation name and e a boolean combination of selection conditions involving R's attributes and constants. The selection conditions involving *temporal attribute(s)* of R may, in addition, use a special constant symbol *now* that is replaced with the *current time*, the time instant referring to the last element of the current history. The ρ and κ symbols indicate whether the tuples satisfying the selection condition e are to be removed from the history or kept while the remaining tuples are kept or removed, respectively.

Similarly to the above simple cases, it is not possible to (statically) detect if a first-order query accesses removed parts of a history and the approach resorts to *run-time* detection of such illegal data accesses.

In addition to introducing policy-based expiration operators, Skyt, et al, [41] also studies situations in which the expiration policies change (or, more precisely are allowed to change). Clearly, once we physically remove parts of the history, we cannot reverse this process to re-obtain the deleted data. Therefore, should we require a change to the expiration policy, we face the question of deciding whether such a change is consistent with the current state of the residual history. In general, policies based on temporal selections from the original history can be changed, but in some cases we may need to *wait* for the new policy to be fully consistent with the residual history.

Example 7. Consider a now-relative keep specification $\kappa(R) : t >$ now $-$ 5. This specification keeps the last five states in the residual history (and removes all other states). To evolve this policy to a new keep specification that keeps last 10 states, $\kappa(R) : t >$ now $- 10$, we need to wait for five time units *without expiring any states from the history* until the new policy takes hold. In the transitional period, the *new* policy is used to expire the data while the old one to determine which queries can be faithfully answered.

4   Query-driven Approaches to Data Expiration

An alternative to a policy based expiration is the removal of unnecessary data based on queries that can be asked over the history. We focus on techniques

handling a fixed set of queries perhaps tailored to the needs of a particular application. Again, in the terminology of materialized view maintenance, we are trying to define special-purpose self-maintainable materialized views that can be used to answer the original queries, but take (much) less space than the original history[5].

For simplicity, all the techniques are introduced with respect to a single query; multiple queries are handled using natural generalizations of the techniques, e.g., by forming a single query whose answers represent the cartesian product or the disjoint union of the results of the individual queries.

Even in this restricted setting, the first question we have to ask ourselves is, whether there is an *optimal expiration operator* for a given query, that is an expiration operator that minimizes the size of $\mathcal{E}(H)$. However, it is easy to see that such an operator cannot be devised for any sufficiently expressive query language.

Proposition 3. *Let L be a query language for which query emptiness is not decidable. Then there cannot be an optimal expiration operator for L.*

This is a consequence of the observation that, for unsatisfiable queries, an optimal expiration operator removes all data from the encoded history. Therefore, we are mainly concerned with expiration operators that *approximate* an optimal operator as well as possible. Intuitively, there are two principal situations in which a value in the history is no longer needed:

1. the value cannot match any selection or join condition in the given query *in any possible extension of the current history*. Thus this value can not *contribute* to the answering of the query and can be removed.
2. the value contributes to answering the query but there is another value that provides *the same answer*; thus one of these values is redundant and can be removed as well.

There are many proposals that expire parts of the history based on the first observation. These techniques [49,50] are commonly based on a *garbage collection-style* reachability analysis [27] of the data (implemented by various mark-and-sweep algorithms).

An important observation at this point is, however, that even (1) alone can only be *approximated* by a computable expiration technique. On the other hand, to achieve a *bounded encoding*, the approximation used has to be tight enough: while most of the current techniques concentrate on (1) we show that (2) is the crux to obtain a bounded encoding.

Example 8. Consider the query (in a hypothetical temporal query language) that asks for all the valuations for tuple of variables $\mathbf{x}$ that have ever appeared in a history (in temporal relational calculus expressed as $\{\mathbf{x} : \exists t.R(t, \mathbf{x})\}$). Evaluating this query accesses *every state* of H.

---

[5] We sidestep the issues of the actual physical representation of relations and assume that the cardinality of the relations is directly proportional to the space needed for their physical representation.

Therefore, *reachability analysis*, that forms the basis of most data expiration methods and that is sound for arbitrary query languages does not provide bounded encoding of histories in the relational setting.

We now turn to the main technical development of this chapter. We introduce two major approaches to data expiration that provide guarantees with respect to the size of the residual data.

4.1   First-order Temporal Logic (FOTL)

Linear-time temporal logic enjoys a widespread popularity in many areas of Computer Science, ranging from temporal query languages and integrity constraint enforcement to model checking of program properties. There are two standard fragments of temporal logic

- Past Temporal Logic, the fragment that uses the ● (in the previous state) and **since** operators only. In this fragment, queries can refer only to the past; therefore the top-level evaluation point is usually set to be the last (current) point in the history.
- Future Temporal Logic, the fragment that uses the ○ (in the next state) and **until** only. Here, the situation is reversed: queries refer only to the future and thus the top-level evaluation point is set to 0 (the first state of the history).

We first focus on the past fragment.

Past Temporal Logic and Materialized Views. We use the following BNF to specify first-order Past Temporal logic (PastTL) queries:

$$Q ::= R(\mathbf{x}) \,|\, F \,|\, Q \wedge Q \,|\, \neg Q \,|\, \exists x.Q \,|\, \bullet Q \,|\, Q \textbf{ since } Q$$

where R is a relational symbol, $\mathbf{x}$ is a tuple of variables, and F is of the form $x = y$. We require the queries to obey the standard syntactic safety rules: all variables must appear as an argument of $R(\mathbf{x})$ or be equated to another such variable, and free variables of subqueries involved in disjunction or negation must match. We also assume that the quantified variables have unique names different from all other variables in the query.

The semantics of queries is defined using the usual satisfaction relation $\models$ that links temporal logic queries with histories (H), substitutions (θ), and an *evaluation point* t.

$$
\begin{aligned}
&H, \theta, t \models R(\mathbf{x}) &&\text{if } \theta(\mathbf{x}) \in \mathbf{R}^{D_t} \text{ for } D_t \in H \\
&H, \theta, t \models x_i = x_j &&\text{if } \theta(x_i) = \theta(x_j) \\
&H, \theta, t \models Q_1 \wedge Q_2 &&\text{if } H, \theta, t \models Q_1 \text{ and } H, \theta, t \models Q_2 \\
&H, \theta, t \models \neg Q &&\text{if not } H, \theta, t \models Q \\
&H, \theta, t \models \exists x_i.Q &&\text{if there is } a \in \mathbf{D}_H \text{ such that } H, \theta[x_i \mapsto a], t \models Q \\
&H, \theta, t \models \bullet Q &&\text{if } t - 1 \in \mathbf{T}_H \text{ and } H, \theta, t - 1 \models Q \\
&H, \theta, t \models Q_1 \textbf{ since } Q_2 &&\text{if } \exists t' \in \mathbf{T}_H.t' < t \wedge H, \theta, t' \models Q_2 \text{ and} \\
&&&\qquad \forall t'' \in \mathbf{T}_H.t' < t'' \leq t \to H, \theta, t'' \models Q_1
\end{aligned}
$$

An answer to a PastTL query Q with free variables $x_1, \ldots, x_k$ with respect to a history H is the relation

$$\{(a_1, \ldots, a_k) \in \mathbf{D}_H^k : H, [x_1/a_1, \ldots, x_k/a_k], n \models Q\},$$

where n is the time instant associated with the *last state* in H. Note that all the temporal variables in the semantic definitions are *restricted to the temporal domain of the history* $\mathbf{T}_H$. Thus, for finite histories, $\bullet Q$ is false in 0-th state of H independently of Q. Additional standard temporal connectives $\blacklozenge$ (sometime in the past) and $\blacksquare$ (always in the past) can be defined in terms of the **since** connective as follows:

$$\blacklozenge X_1 \overset{\triangle}{=} \text{true } \mathbf{since } X_1 \qquad \blacksquare X_1 \overset{\triangle}{=} \neg \blacklozenge \neg X_1$$

Example 9. Students who have TA'ed at least one class twice in the past.

$$\{x : \blacklozenge(\exists y.\mathrm{TA}(x, y) \land \blacklozenge \mathrm{TA}(x, y))\}$$

The popularity of temporal logic can often be traced to the fact that temporal connectives can be defined inductively over a history (or in general a temporal structure) by the following equivalence:

$$Q_1 \mathbf{ since } Q_2 \equiv Q_1 \land (\bullet Q_2 \lor \bullet(Q_1 \mathbf{ since } Q_2))$$

In particular, this equivalence is the key for the link between propositional temporal logics and automata theory. Also, together with the observation that in the 0-th state of a history $\bullet Q$ is identically false, the equivalence yields an alternative inductive definition of the satisfaction relation with respect to a history.

This approach was developed as a practical method for checking temporal logic constraints formulated in PastTL formulas [10,12]. However, the approach directly extends to answering (a fixed set of) PastTL queries with respect to the current (last) point of a history H.

The method is based on the equivalence above and works as follows: given a PastTL query Q we define an *auxiliary relation* R_α for every temporal subformula α of Q. The arities of these relations are defined by the number of free variables in the corresponding temporal subformulas. We denote $Q[\alpha/R_\alpha]$ a modification of the query Q in which the subformula α has been substituted by R_α.

Definition 2. Let $R_1, \ldots, R_k$ be a relational schema of states in a history H and Q a PastTL query over H. We define an expiration operator $\mathcal{E}_{past}$ for a given query Q that maps a history H to a *relational database* instance with the schema $R_1, \ldots, R_k, R_{\alpha_1}, \ldots, R_{\alpha_l}$ defined inductively with respect to extensions of H as follows:

$$0^{\mathcal{E}_{past}} = (\mathbf{R}_1^0, \ldots, \mathbf{R}_k^0, \emptyset, \ldots, \emptyset),$$

$$\Delta^{\mathcal{E}_{past}}((\mathbf{R}_1^{n-1},\ldots,\mathbf{R}_k^{n-1},\mathbf{R}_{\alpha_1}^{n-1},\ldots,\mathbf{R}_{\alpha_l}^{n-1}),(\mathbf{R}_1^n,\ldots,\mathbf{R}_k^n)) =$$
$$(\mathbf{R}_1^n,\ldots,\mathbf{R}_k^n,Q_{\alpha_1}^n,\ldots,Q_{\alpha_l}^n).$$

The queries Q_α that define the new state for the auxiliary materialized views R_α are defined as follows:

$$Q_\alpha^n = \begin{cases} Q^{n-1} & \text{for } \alpha = \bullet Q \\ Q_1^n \wedge (Q_2^{n-1} \vee R_\alpha^{n-1}) & \text{for } \alpha = Q_1 \text{ since } Q_2 \end{cases}$$

where Q^i denotes evaluating the query $Q[\alpha/R_\alpha]$ in the state i. To complete the definition of the expiration operator we define

$$Q^{\mathcal{E}_{past}} = Q[\alpha_i/R_{\alpha_i}],$$

and $\mathcal{E}_{past}(Q) = (0^{\mathcal{E}_{past}}, \Delta^{\mathcal{E}_{past}}, Q^{\mathcal{E}_{past}})$.

Since all temporal subformulas of Q have been replaced by the auxiliary views (both in $Q^{\mathcal{E}_{past}}$ and in the inductive definitions of the auxiliary views Q_α), evaluating Q^n now only needs access to the states n (the current state) and $n-1$ (the last state).

Every time a new state S is being added to a history H, the inductive definitions of the auxiliary relations R_α allow us to refresh their instances based on the last state of H and S. Thus, at each point of time, it is sufficient to keep at most two states of the history.

The expiration operator is thus defined to keep the last state of the *extended* history while incrementally maintaining the content of the auxiliary relations using the inductive definitions.

Example 10. The PastTL query in Example 9 contains two temporal subqueries,

$\alpha_2 = \blacklozenge \text{TA}(x,y)$ and
$\alpha_1 = \blacklozenge \exists y.\text{TA}(x,y) \wedge \blacklozenge \text{TA}(x,y)$.

Therefore, the extended database state contains two auxiliary views, R_{α_1} (a binary relation) and R_{α_2} (an unary relation). Their content is inductively defined using the above rules. Applying these rules on our example history (cf. Example 6) yields the following sequence:

	$R_{\alpha_1}(x,y)$	$R_{\alpha_2}(x)$
0	{}	{}
1	{(John, CS448)}	{}
2	{(John, CS448), (Sue, CS234)}	{John}
3	{(John, CS448), (Sue, CS234)}	{John}

Space Bounds The above observation immediately provides an upper bound on the size of $\mathcal{E}(H)$: $\mathcal{E}(H)$ is a standard *relational database* over the original schema extended with a fixed set of auxiliary views (with fixed arity). Therefore, $|\mathcal{E}(H)| \leq O(|\mathbf{D}_H|^k)$ where k is the maximal arity in the extended schema. In particular, this result allows $|\mathcal{E}(H)|$ to be at most *polynomially larger* than $|H|$.

Example 11. Consider the query

$$\blacklozenge(p(x_1) \wedge \ldots \wedge p(x_k))$$

and a history

$$H = \langle \{a_1\}, \{a_2\}, \{a_3\}, \ldots, \{a_n\} \rangle.$$

Using the method defined above the size of the materialized view R_α is

$$|R_\alpha| = (n-1)^k$$

for $\alpha = \blacklozenge(p(x_1) \wedge \ldots \wedge p(x_k))$ with free variables $x_1, \ldots, x_k$ with respect to the history H of size n. In fact the same holds for every prefix of H.

In summary, the above construction yields:

Proposition 4. *There is an expiration operator $\mathcal{E}_{past}$ for PastTL that guarantees bounded encoding of histories.*

The approach has been implemented using the *materialized view* technology on top of an *active DBMS*; database *triggers* were used to maintain the content of the materialized views implementing the extended state [12].

Fixpoint Extensions of Past Temporal Logic. The materialized view-based method for PastTL can be extended to handle fixpoint (more precisely PastμTL) formulas as well. A PastμTL formula is defined by the grammar

$$Q ::= R(\mathbf{x}) \,|\, F \,|\, Q \wedge Q \,|\, \neg Q \,|\, \exists x.Q \,|\, \bullet Q \,|\, \mu X.Q.$$

PastμTL extends the first-order PastTL with a *fixpoint operator* $\mu X.Q$ where X is a relational *variable* standing for a relation of the same arity as Q. The semantics of PastμTL is defined the same way as the semantics of PastTL; the $\mu X.Q$ *least fixpoint operator* is interpreted as follows:

$$H, \theta, i \models \mu X.Q \text{ if } H', \theta, i \models Q$$

where H' is a copy of the history H in which each state is extended with an interpretation X in such a way that this extension is the least solution of $X \iff Q(X)$ [47]. A special case of a PastμTL query is a guarded PastμTL query, a query in which all occurrences of the $\mu X.Q$ are of the form $\mu X.Q(\bullet X)$. In particular, the standard past temporal operator **since** can be defined using the following PastμTL definition.

Example 12. The **since** connective can be defined by the following guarded fixpoint formula:

$$Q_1 \textbf{ since } Q_2 = \mu X.Q_1 \wedge (\bullet Q_2 \vee \bullet X);$$

this definition exactly mirrors the *unfolding rule* fir the **since** connective introduced introduced in the beginning of this section.

The definition of an *expiration operator* for a PastμTL formula is based on the *unfolding property* of the fixpoint operation:

$$\mu X.Q \equiv Q(\mu X.Q)$$

Similarly to the (first-order) Past Temporal logic, we define auxiliary relations for temporal subformulas of the given PastμTL query to define an expiration operator. The only difference from Definition 2 lies in the way the auxiliary materialized views are recomputed after a new state is appended to the history. The queries that define the view refresh are defined as follows:

$$Q_\alpha^n = \begin{cases} Q^{n-1} & \text{for } \alpha = \bullet Q \\ Q^n & \text{for } \alpha = \mu X.Q \end{cases}$$

Proposition 5. *There is an expiration operator $\mathcal{E}_\mu$ for PastμTL that guarantees bounded encoding of histories. Moreover, for guarded PastμTL formulas, $\Delta^{\mathcal{E}_\mu}$ and $Q^{\mathcal{E}_\mu}$ are defined using first-order queries.*

The fact that for guarded PastμTL formulas, the queries that define the new instances of the materialized views are first-order queries follows immediately from the definition of the Q_α formulas: since all temporal subformulas are replaced by auxiliary predicate symbols the fixpoints are not needed because the subformulas in the scope of the fixpoint operator no longer contain the fixpoint variable. The space bounds for PastμTL expiration operator mirror those obtained for PastTL. PastμTL, however, allows queries that cannot be formulated in PastTL [48].

Example 13. Consider the query

$$\mu X.\exists y.\mathrm{TA}(x, y) \vee \bullet \bullet X$$

that lists all students who TA'ed "in all the even" terms (counting back from the current term and assuming two terms per year). The temporal subformulas of this query are the following: $\alpha_1 = \mu X.\exists y.\mathrm{TA}(x, y) \vee \bullet \bullet X$, $\alpha_2 = \bullet \bullet X$, and $\alpha_3 = \bullet X$. The inductive definitions for auxiliary relations R_{α_1}, R_{α_2}, and R_{α_3}, when applied on our example history, yield the following:

	$R_{\alpha_1}(x)$	$R_{\alpha_2}(x)$	$R_{\alpha_3}(x)$
0	{John}	{}	{}
1	{John, Sue}	{}	{John}
2	{John}	{John}	{John, Sue}
3	{John, Sue}	{John, Sue}	{John}

Intuitively, R_{α_2} and R_{α_3} model the odd and even states, respectively while R_{α_1} adds the facts found in the history to the appropriate set.

Metric PastTL and Real-time Queries. Another extension of PastTL, called *Past Metric TL* [4,28], allows temporal operators that refer the *real time instant* associated with the particular state of the history. This extension assumes a *more structured* model of time, in particular, a model that captures *temporal distance* between consecutive states of a history (*duration*).

The extension is realized by adding the connectives $\bullet_{\sim c}$ and $\mathbf{since}_{\sim c}$, where c is a non negative integer (standing for *duration*) and $\sim \in \{<, =, >\}$, are added to the PastTL language. The semantics of these new connectives (conceptually) utilizes an additional constant clk that satisfies

$$H, [x = t], i \models \mathrm{clk} = x \text{ if } t \text{ is the } \textit{real time} \text{ instant associated with } D_i.$$

We require that the value *clk* increases as the history grows and that the clocks in the history satisfy

$$\exists \epsilon > 0 \forall i \in N : \mathrm{clk}^{i+1} - \mathrm{clk}^i > \epsilon.$$

(for discrete time $\epsilon = 1$). We denote clk^i the value of the clock at state i.

The semantics is an extension of the semantics of the standard PastTL connectives and is defined as follows:

$$
\begin{aligned}
H, \theta, t \models \bullet_{\sim c} Q \qquad & \text{if} \quad t - 1 \in \mathbf{T}_H, H, \theta, t - 1 \models Q \text{ and} \\
& \qquad \mathrm{clk}^t - \mathrm{clk}^{t-1} \sim c \\
H, \theta, t \models Q_1 \, \mathbf{since}_{\sim c} \, Q_2 \quad & \text{if} \quad \exists t' \in \mathbf{T}_H.t' < t, H, \theta, t' \models Q_2, \\
& \qquad \mathrm{clk}^t - \mathrm{clk}^{t'} \sim c, \text{ and} \\
& \qquad \forall t'' \in \mathbf{T}_H.t' < t'' \le t \to H, \theta, t'' \models Q_1
\end{aligned}
$$

The main difference between PastTL and Metric PastTL lies in the fact that the states involved in the temporal operators must, in addition to being *ordered* according to the requirements of the temporal operator, obey certain *distance* constraints. More specifically,

- the clocks associated with the consecutive time instants related by the $\bullet$ connective must obey the $\sim c$ constraint (note that the *next state* is still defined the same way as in pure PastTL).
- the clock associated with the state of Q_2 in $Q_1 \, \mathbf{since}_{\sim c} \, Q_2$ must be related to the clock of the *current state* in a similar fashion.

The first requirement can be enforced simply by adding the additional constraint to the incremental definition of the auxiliary relation associated with the $\bullet Q$ subformulas. The modification handling the subformulas of the form $\alpha = Q_1 \, \mathbf{since}_{\sim c} \, Q_2$ is more complex. In order to enforce the $\sim c$ constraint we associate with each tuple in the auxiliary table R_α a value $d \in \{0, \ldots, c + 1\}$

that records the *distance in the past* of the particular tuple (i.e., how far in the past has this tuple occurred in $Q_2(H)$; the value $c + 1$ stands for all distances beyond c). Using this additional value we can define the incremental rules for maintaining R_α as follows:

$$R_\alpha^0(\mathbf{x}, d) = \text{false}$$
$$R_\alpha^n(\mathbf{x}, d) = (Q_1^n \wedge ((Q_2^{n-1} \wedge d' = 0) \vee R_\alpha^{n-1}(\mathbf{x}, d')) \wedge$$
$$d = \min\{c + 1, d' + \text{clk}^n - \text{clk}^{n-1}\}$$

The replacement for a temporal subformula α of the form $Q_1 \textbf{ since}_{\sim c} Q_2$ in queries is then the expression $\exists d.R_\alpha(\mathbf{x}, d) \wedge d \sim c$. The definitions are patterned after rules used for enforcement of integrity constraints formulated in Metric PastTL [9]. These definitions are used to define a bounded expiration operator for Metric PastTL. The crucial observation is that the d values stored in the auxiliary table R_α range only over a finite set of values that is independent of the length of the history.

The condition forcing the clocks to advance by at least a fixed $\epsilon > 0$ is essential to obtain a bounded encoding.

Example 14. Consider a history H in which $\text{clk}^i = 1 - 1/2^i$. In this history, the cardinality of the set of d values in the auxiliary relation R_α can grow linearly with the length of the history.

Future Temporal Logic and Finite Automata. The other commonly explored option considers the *future* fragment of temporal logic. The following BNF defines the syntax of FutureTL:

$$Q ::= R(\mathbf{x}) \,|\, F \,|\, Q \wedge Q \,|\, \neg Q \,|\, \exists x.Q \,|\, \bigcirc Q \,|\, Q \textbf{ until } Q$$

The semantics of queries is defined similarly to PastTL, the only difference is in the definitions of the temporal operators:

$$H, \theta, t \models \bigcirc Q \qquad \text{if } t + 1 \in \mathbf{T}_H \text{ and } H, \theta, t + 1 \models Q$$
$$H, \theta, t \models Q_1 \textbf{ until } Q_2 \text{ if } \exists t' \in \mathbf{T}_H.t < t' \wedge H, \theta, t' \models Q_2 \text{ and}$$
$$\forall t'' \in \mathbf{T}_H.t < t'' \leq t' \rightarrow H, \theta, t'' \models Q_1$$

Note again that all the temporal variables in the semantic definitions are *restricted to the temporal domain of the history* $\mathbf{T}_H$. Similar to the case of past fragments, additional temporal connectives can be defined in terms of **until** as follows:

$$\diamond X_1 \triangleq \text{true } \textbf{until } X_1 \qquad \square X_1 \triangleq \neg \diamond \neg X_1$$

In contrast with PastTL, queries in the future fragment are evaluated with respect to the *first* time instant in the history, 0. The following equivalence holds for the **until** operator:

$$Q_1 \textbf{ until } Q_2 \equiv Q_1 \wedge (\bigcirc Q_2 \vee \bigcirc(Q_1 \textbf{ until } Q_2))$$

The above equivalence suggests the possibility of using the connection between (infinite) histories that are models of FutureTL formulas and strings accepted by an (finite, Büchi) automaton to construct a finite (and bounded) representation of an expiration operator for FutureTL. Indeed, the connection for *propositional* FutureTL is well known [39,36,46] and has been successfully used in the area of Model checking (cf. Section 6).

However, extending these results directly to *first-order* FutureTL is not possible. First, unlike the propositional FutureTL, the first-order extension satisfiability of sentences with respect to infinite histories is not decidable (cf. Section 5 for details). Thus, in our setting, we consider only finite prefixes of histories, known at each time. The automata construction, in particular, the acceptance conditions, can be adjusted to accepting finite histories that satisfy a propositional FutureTL formula [35].

However, the main obstacle in developing this approach is the handling of *data quantifiers* in the first-order "*data*" part of the query language. The difficulty we face is that data quantifiers do not distribute/commute with temporal connectives. This way the quantifiers can introduce *new* data variables in multiple temporal contexts (e.g., when the **until** connective is *unfolded*). These new variables then lead to creating a potentially infinite number of states, one for each **until** unfolding, in the standard automaton construction. Indeed, we show in Section 4.3 that, for fixpoint extensions of FutureTL, the automata-based technique necessarily fails to produce a bounded expiration operator due to data quantification.

Lipeck and Saake and others [34,24] attempt to avoid this problem by restricting temporal formulas to *biquantified formulas*: those in which temporal operators do not appear in the scope of data quantifiers (with the exception of the final universal closure applied to the remaining free variables). For these formulas the standard propositional construction of a finite automaton accepting a state sequence can be used. A final touch is needed to capture the sets of substitutions for the free data variables. Hülsmann and Saake [24] use a *constraint representation* [30] of sets of answer substitutions to construct the final answer. This technique, however, has its advantages: the automaton (called transition graph, [18]) can be converted to a set of triggers in an active database system.

Alternatively, in situations in which unrestricted FutureTL queries are needed, we obtain a bounded expiration operator by embedding a FutureTL query into temporal relational calculus and using techniques introduced in next section.

4.2   Temporal Relational Calculus (2-FOL)

The second approach to temporalizing relational calculus introduces explicit references to time instants (variables) and the associated linear ordering

of time and quantifiers over time. We use the standard syntax for range-restricted first-order queries.

$$Q ::= R(t, \mathbf{x}) \mid F \mid Q \wedge Q \mid \exists x.Q \mid \exists t.Q \mid Q \wedge \neg Q \mid Q \vee Q$$

where R is a relational symbol, $\mathbf{x}$ is a tuple of variables, and F is of the form $x = y$ for data variables and $t < s$ for temporal variables. Similarly to PastTL formulas, we require queries to obey the standard syntactic safety rules. The semantics of Temporal Relational Calculus queries is defined as follows:

$$
\begin{array}{lll}
H, \theta \models R(t, \mathbf{x}) & \text{if} & \mathbf{x}\theta \in \mathbf{R}^{D_{t\theta}} \\
H, \theta \models t_i < t_j & \text{if} & \theta(t_i) < \theta(t_j) \\
H, \theta \models x_i = x_j & \text{if} & \theta(x_i) = \theta(x_j) \\
H, \theta \models Q_1 \wedge Q_2 & \text{if} & H, \theta \models Q_1 \text{ and } H, \theta \models Q_2 \\
H, \theta \models \neg Q & \text{if} & \text{not } H, \theta \models Q \\
H, \theta \models \exists t_i.Q & \text{if} & \text{there is } s \in \mathbf{T}_H \text{ such that } H, \theta[t_i \mapsto s] \models Q \\
H, \theta \models \exists x_i.Q & \text{if} & \text{there is } a \in \mathbf{D}_H \text{ such that } H, \theta[x_i \mapsto a] \models Q
\end{array}
$$

The only difference from the standard definition of the satisfaction relation is in the case of base relations: the base relations are evaluated at the point of the history specified by their first argument. We assume that the valuations θ always map variables to values of the appropriate domain and are restricted to the free variables of the particular query.

Example 15. Students who have taken at least one class twice.

$$\{x : \exists t_1, t_2.t_1 < t_2 \wedge \exists y.\text{TA}(t_1, x, y) \wedge \text{TA}(t_2, x, y)\}$$

Note that the binary relation TA (binary in every state of a history) corresponds to a *ternary* predicate in temporal relational calculus: the first argument of the predicate indicates the time instant in the history at which the relation TA is considered [13].

Materialized views have proven to be very successful approach for expiring histories with respect to temporal logic queries. However, there is little hope such techniques can work for general first-order queries, a language strictly stronger than temporal logic.

Proposition 6 ([1,45]). *Temporal logic is strictly weaker than first-order logic.*

The separation can be traced to the fact that first-order logic can reference *multiple temporal contexts* in a query, while temporal logic always references a *single temporal context*.

Explicit access to temporal contexts involves variables ranging over time instants. The following example shows a simple case where the ability to retrieve temporal values immediately precludes any possibility of bounded history encoding.

Example 16. Consider a query $R(t, \mathbf{x})$ for $R \in \rho$. The correct answer consists of the whole history of R in the warehouse; therefore we need to know (and thus store) the whole history to compute the answer for the above query.

This problem, however, can be avoided by *restricting* the query language to queries with *bounded answers*: $|Q(H)| \leq f(|\mathbf{D}_H|)$ (cf. Example 8). Unfortunately, due to the usual *query emptiness* reduction we have:

Theorem 1. *Whether a query is bounded is undecidable.*

In the rest of this section we therefore restrict our attention to queries that are *syntactically* bounded: we require all free variables in the query to be *data* variables.

The proposed solution is based on two techniques: first we *specialize* a given query with respect to the known part of the history to detect values that can be removed and second, we extract a *residual history* from the specialized query.

Specialization of Queries The partial-evaluation technique is based on treating relations in the known history H and in all its possible extensions H' as *characteristic* formulas based on equality and order constraints. Given a history H we define *abstract substitutions* to be the formulas

$$[{}^x_a] \equiv \begin{cases} x = a & a \in \mathbf{D}_H \\ \forall a \in \mathbf{D}_H . x \neq a & a = \bullet \end{cases}$$

and

$$[{}^t_s] \equiv \begin{cases} t = s & s \in \mathbf{T}_H \\ t > \mathsf{Max}(\mathbf{T}_H) & s = \bullet \end{cases}$$

where x is a data variable, t a temporal variable and $\bullet \notin \mathbf{D}_H \cup \mathbf{T}_H$ a new symbol; this symbol is used to denote all the values outside of the (current) active data and temporal domains. We allow composite abstract substitutions to denote a finite conjunction of the above formulas, e.g., $[{}^{xy}_{ab}]$ denotes the conjunction of $[{}^x_a]$ and $[{}^y_b]$.

Note that different abstract substitutions always denote disjoint sets. The definition allows us to treat relations as disjunctions of abstract substitutions. In particular, we can substitute these disjunctions for the leaves of the original query and then use the usual simplification rules for first-order formulas to specialize the query:

Definition 3 (Query Specialization). Let H be a history. We define a function PE_H that maps a query Q to a set of *residual queries* indexed by *abstract substitutions*, $Q'[{}^{\mathbf{x}}_{\mathbf{a}}]$, where $\mathbf{x}$ is the set of free variables of Q and $\mathbf{a}$ is the corresponding set of abstract values (of the appropriate type). The function PE_H is defined inductively on the structure of Q in Figure 1.

This definition yields the following specialized query when applied to our running example history (Example 6) and query (Example 15):

$$\mathsf{PE}_H(Q) = \begin{cases}
\{\text{true}[^{t\mathbf{x}}_{s\mathbf{a}}] : R(s,\mathbf{a}) \in H\} \\
\quad \cup \{R(t,\mathbf{x})[^{t\mathbf{x}}_{\bullet\mathbf{a}}] : \mathbf{a} \in (\mathbf{D}\cup\{\bullet\})^{|\mathbf{x}|}\} & Q \equiv R(t,\mathbf{x}) \\
\{Q'_1[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_1), [^{\mathbf{x}}_{\mathbf{a}}] \wedge F \text{ is satisfiable}\} & Q \equiv Q_1 \wedge F \\
\{Q'_1 \wedge Q'_2[^{\mathbf{xy}}_{\mathbf{ab}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_1), \\
\quad\quad Q'_2[^{\mathbf{y}}_{\mathbf{b}}] \in \mathsf{PE}_H(Q_2), \models [^{\mathbf{xy}}_{\mathbf{ab}}]\} & Q \equiv Q_1 \wedge Q_2 \\
\{(\exists y. \bigvee_{Q'_1[^{\mathbf{xy}}_{\mathbf{ab}}]\in\mathsf{PE}_H(Q_1)} Q'_1)[^{\mathbf{x}}_{\mathbf{a}}] : \exists b.Q''_1[^{\mathbf{xy}}_{\mathbf{ab}}] \in \mathsf{PE}_H(Q_1)\} & Q \equiv \exists y.Q_1 \\
\{(\exists t. \bigvee_{Q'_1[^{\mathbf{x}t}_{\mathbf{a}s}]\in\mathsf{PE}_H(Q_1)} Q'_1)[^{\mathbf{x}}_{\mathbf{a}}] : \exists s.Q''_1[^{\mathbf{x}t}_{\mathbf{a}s}] \in \mathsf{PE}_H(Q_1)\} & Q \equiv \exists t.Q_1 \\
\{Q'_1 \wedge \neg Q'_2[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_1), Q'_2[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_2)\} \\
\quad \cup \{Q'_1[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_1), Q'_2[^{\mathbf{x}}_{\mathbf{a}}] \notin \mathsf{PE}_H(Q_2)\} & Q \equiv Q_1 \wedge \neg Q_2 \\
\{Q'_1 \vee Q'_2[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1 \in \mathsf{PE}_H(Q_1)[^{\mathbf{x}}_{\mathbf{a}}], Q'_2[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_2)\} \\
\quad \cup \{Q'_1[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_1), Q'_2[^{\mathbf{x}}_{\mathbf{a}}] \notin \mathsf{PE}_H(Q_2)\} \\
\quad \cup \{Q'_2[^{\mathbf{x}}_{\mathbf{a}}] : Q'_1[^{\mathbf{x}}_{\mathbf{a}}] \notin \mathsf{PE}_H(Q_1), Q'_2[^{\mathbf{x}}_{\mathbf{a}}] \in \mathsf{PE}_H(Q_2)\} & Q \equiv Q_1 \vee Q_2
\end{cases}$$

Fig. 1. Query Specialization.

Example 17. The PE_H operator applied on the subquery

$$t_1 < t_2 \wedge \exists y.\text{TA}(t_1, x, y) \wedge \text{TA}(t_2, x, y)$$

yields the following set of formulas:

$$\begin{array}{ll}
 & \text{true}\,[^{t_1 t_2 x}_{0\ 1\ \text{John}}] \\
 & \text{true}\,[^{t_1 t_2 x}_{0\ 2\ \text{John}}] \\
 & \text{true}\,[^{t_1 t_2 x}_{1\ 2\ \text{John}}] & \text{true}\,[^{t_1 t_2 x}_{1\ 3\ \text{Sue}}] \\
\exists y.(\text{TA}(t_2, x, y)[^{y}_{\text{CS448}}])\,[^{t_1 t_2 x}_{0\ \bullet\ \text{John}}] & \exists y.(\text{TA}(t_2, x, y)[^{y}_{\text{CS234}}])\,[^{t_1 t_2 x}_{1\ \bullet\ \text{Sue}}] \\
\exists y.(\text{TA}(t_2, x, y)[^{y}_{\text{CS448}}])\,[^{t_1 t_2 x}_{1\ \bullet\ \text{John}}] & \exists y.(\text{TA}(t_2, x, y)[^{y}_{\text{CS234}}])\,[^{t_1 t_2 x}_{3\ \bullet\ \text{Sue}}] \\
\exists y.(\text{TA}(t_2, x, y)[^{y}_{\text{CS448}}])\,[^{t_1 t_2 x}_{2\ \bullet\ \text{John}}]
\end{array}$$

(the formulas with abstract substitutions of the form $[^{t_1 t_2 x}_{\bullet\ \bullet\ a}]$ for all possible $a \in \mathbf{D}_H \cup \{\bullet\}$ omitted for brevity). For example, the first specialized formula, $\text{true}[^{t_1 t_2 x}_{0\ 1\ \text{John}}]$, is derived from the first two states of the history that assert that John was a TA for the CS448 class in the 0th and 1st terms. Similarly, the 4th specialized formula is derived since John has been a TA for CS448 in the 0th term and he might be the TA for the class again when the history is extended.

The specialization itself has little impact on the size of the residual query $\mathsf{PE}_H(Q)$, it merely eliminates inconsistent conjunctions from the residual formula. As we pointed out in Example 8, this is not sufficient to obtain a bounded encoding for H. However, for the specialized queries (and their subqueries) we can define an equivalence relation that identifies subformulas that *behave in the same way in all extensions of H*. The equivalence relation,

$Q = R$: Let $Q_1[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}], Q_2[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \in \mathsf{PE}_H\, Q$. Then

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff ((Q_1 = Q_2 = \text{ true}) \vee (\mathbf{a}_1 = \mathbf{a}_2)),$$

$Q = Q' \wedge F$:

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff \left([\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \text{ where } [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \wedge F \text{ and } [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \wedge F \text{ are satisfiable}\right),$$

$Q = \exists y.Q'$: Let $S_1 = \{b : Q_1'[\begin{smallmatrix}y\mathbf{x}\\b\mathbf{a}_1\end{smallmatrix}] \in \mathsf{PE}_H(Q')\}$ and $S_2 = \{b : Q_2'[\begin{smallmatrix}y\mathbf{x}\\b\mathbf{a}_2\end{smallmatrix}] \in \mathsf{PE}_H(Q')\}$. Then

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff \left(\begin{array}{r}(\forall b \in S_1 \exists c \in S_2.[\begin{smallmatrix}y\mathbf{x}\\b\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}y\mathbf{x}\\c\mathbf{a}_2\end{smallmatrix}]) \\ \wedge\, (\forall c \in S_2 \exists b \in S_1.[\begin{smallmatrix}y\mathbf{x}\\b\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}y\mathbf{x}\\c\mathbf{a}_2\end{smallmatrix}])\end{array}\right),$$

$Q = \exists t.Q'$: Let $S_1 = \{s : Q_1'[\begin{smallmatrix}t\mathbf{x}\\s\mathbf{a}_1\end{smallmatrix}] \in \mathsf{PE}_H(Q')\}$ and $S_2 = \{s : Q_2'[\begin{smallmatrix}t\mathbf{x}\\s\mathbf{a}_2\end{smallmatrix}] \in \mathsf{PE}_H(Q')\}$. Then

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff \left(\begin{array}{r}(\forall s \in S_1 \exists u \in S_2.[\begin{smallmatrix}y\mathbf{x}\\s\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}y\mathbf{x}\\u\mathbf{a}_2\end{smallmatrix}]) \\ \wedge\, (\forall u \in S_2 \exists s \in S_1.[\begin{smallmatrix}y\mathbf{x}\\s\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}y\mathbf{x}\\u\mathbf{a}_2\end{smallmatrix}])\end{array}\right),$$

$Q = Q' \wedge Q''$:

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff \left([\begin{smallmatrix}\mathbf{x}'\\\mathbf{a}_1'\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}\mathbf{x}'\\\mathbf{a}_2'\end{smallmatrix}] \wedge [\begin{smallmatrix}\mathbf{x}''\\\mathbf{a}_1''\end{smallmatrix}] \sim_{Q''}^H [\begin{smallmatrix}\mathbf{x}''\\\mathbf{a}_2''\end{smallmatrix}], \text{ for } [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_i\end{smallmatrix}] = [\begin{smallmatrix}\mathbf{x}'\mathbf{x}''\\\mathbf{a}_i'\mathbf{a}_i''\end{smallmatrix}] \text{ satisfiable}\right),$$

$Q = Q' \wedge \neg Q''$ or $Q = Q' \vee Q''$:

$$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \iff \left([\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_{Q'}^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}] \wedge [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_{Q''}^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}]\right).$$

Fig. 2. Equivalence of Abstract Substitutions

$[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}] \sim_Q^H [\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}]$, defined in Figure 2 relates abstract substitutions $[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_1\end{smallmatrix}]$ and $[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}_2\end{smallmatrix}]$ and the specialized queries associated with these substitutions. Intuitively, the equivalence relation asserts that the related abstract substitutions always *behave* the same no matter how the current history is extended. For example, since the history is append-only, the *past* states are always present (and thus are equivalent).

Example 18. Continuing with our running example, the equivalence classes are

$$\left\{[\begin{smallmatrix}t_1 t_2 x\\0\ 1\ \text{John}\end{smallmatrix}], [\begin{smallmatrix}t_1 t_2 x\\0\ 2\ \text{John}\end{smallmatrix}], [\begin{smallmatrix}t_1 t_2 x\\1\ 2\ \text{John}\end{smallmatrix}]\right\}, \left\{[\begin{smallmatrix}t_1 t_2 x\\0\ \bullet\ \text{John}\end{smallmatrix}], [\begin{smallmatrix}t_1 t_2 x\\1\ \bullet\ \text{John}\end{smallmatrix}], [\begin{smallmatrix}t_1 t_2 x\\2\ \bullet\ \text{John}\end{smallmatrix}]\right\},$$

$$\left\{[\begin{smallmatrix}t_1 t_2 x\\1\ 3\ \text{Sue}\end{smallmatrix}]\right\}, \left\{[\begin{smallmatrix}t_1 t_2 x\\1\ \bullet\ \text{Sue}\end{smallmatrix}], [\begin{smallmatrix}t_1 t_2 x\\3\ \bullet\ \text{Sue}\end{smallmatrix}]\right\}$$

(again, the equivalence classes for the remaining substitutions are omitted.)

This equivalence relation allows us to pick a single representative for each equivalence class, in particular for formulas under the existential quantifier. We concentrate on *temporal variables* only as that is sufficient to achieve

bounded encoding for histories. In general we could have used the same approach for data variables as well.

Definition 4 (Time Base). Let Q be a query of the form $\exists t.Q'$ and $[\begin{smallmatrix}\mathbf{x}\\\mathbf{a}\end{smallmatrix}]$ an abstract substitution for free variables of Q. We define a *time base* for t with respect to $\mathbf{a}$ as the set $\mathsf{TB}_{\mathbf{a}}(t) \subseteq \{s : Q'_1[\begin{smallmatrix}\mathbf{x}t\\\mathbf{a}s\end{smallmatrix}] \in \mathsf{PE}_H(Q')\}$ such that

$$\forall s'.Q'_1[\begin{smallmatrix}\mathbf{x}t\\\mathbf{a}s'\end{smallmatrix}] \in \mathsf{PE}_H(Q') \Rightarrow$$
$$\exists s \in \mathsf{TB}_{\mathbf{a}}(t).Q'_2[\begin{smallmatrix}\mathbf{x}t\\\mathbf{a}s\end{smallmatrix}] \in \mathsf{PE}_H(Q') \wedge [\begin{smallmatrix}\mathbf{x}t\\\mathbf{a}s'\end{smallmatrix}] \sim^H_{Q'} [\begin{smallmatrix}\mathbf{x}t\\\mathbf{a}s\end{smallmatrix}]$$

Note that $\mathsf{TB}_{\mathbf{a}}(t)$ has to contain an element for every equivalence class of $\sim^D_{Q'}$.

In other words, $\mathsf{TB}_{\mathbf{a}}(t)$ contains a representative for every equivalence class of $\sim^H_{Q'}$ relation. In practice we can, for example, pick the least value of each equivalence class according to the ordering of time instants.

Example 19. The minimal elements in the equivalence classes define the time base for the variable t_1 as follows:

$$\mathsf{TB}_{1,\text{John}}(t_1) = \{0\}$$
$$\mathsf{TB}_{2,\text{John}}(t_1) = \{0\} \qquad \mathsf{TB}_{3,\text{Sue}}(t_1) = \{1\}$$
$$\mathsf{TB}_{\bullet,\text{John}}(t_1) = \{0\} \qquad \mathsf{TB}_{\bullet,\text{Sue}}(t_1) = \{1\}$$

Therefore, in the expired history, it is sufficient to keep only states 1 and 2 as valuations for the variable t_1.

Using the sets $\mathsf{TB}_{\mathbf{a}}(t)$ we can eliminate superfluous disjuncts under existential quantifiers in the residual formulas of $\mathsf{PE}_H(Q)$. The result of this transformation, $\mathsf{PE}_H(Q)_{|\sim^H}$, can be used directly to encode the history H as follows.

$$Q(H) = \mathsf{PE}_H(Q)_{|\sim^H}(\emptyset)$$
$$\mathsf{PE}_{H;H'}(Q)_{|\sim^H} i \equiv \mathsf{PE}_{H'}(\mathsf{PE}_H(Q)_{|\sim^H})_{|\sim^H}$$

The above approach falls into the category of *query specialization* based techniques for database histories. However, in this paper we look for a *physical deletion* strategy. Therefore we continue to modify the residual query obtained by the PE_H operator as follows:

Definition 5 (Temporal Support). Let t be a temporal variable in Q (i.e., Q contains a subformula $\exists t.Q'$). We define a *temporal support of t* to be the set
$$\mathsf{TB}(t) = \bigcup \mathsf{TB}_{\mathbf{a}}(t)$$

where the union ranges over all remaining abstract substitutions generated by $\mathsf{PE}_H(Q)$ for Q'.

Note that $\mathsf{TB}(t)$ is a valid time base for all abstract substitutions $\mathbf{a}$ (cf. Definition 4). This definition restricts the valuations of t to the set of *non-redundant* valuations $\mathsf{TB}(t)$. Therefore every quantifier of the form $\exists t.Q'$ in $\mathsf{PE}_H(Q)$ can be replaced by a restricted quantifier $\exists t \in \mathsf{TB}(t).Q'$. This formula, however, is equivalent to the original query Q in which the quantifiers are bounded in the same way. Thus we can simply consider these sets, restricted to the current active domain, to be *auxiliary* 0-ary (in the data dimension) relations added to D.

In this setting we are ready to define the actual expiration operator for a fixed query Q as follows.

- $Q^{\mathcal{E}}$ is identical to Q except all subformulas of the form $\exists t_j.Q'$ are replaced by $\exists t_j.\mathsf{TB}_{t_j} \wedge Q'$ where TB_{t_j} are additional 0-ary auxiliary relations.
- $0^{\mathcal{E}} = \langle\,\rangle$, and
- $\Delta^{\mathcal{E}}(\mathcal{E}(H), H')$ is defined as the history

$$\langle D_i : D_i \in \mathcal{E}(H); H', \ i \in \mathsf{TB}(t_j) \text{ for some variable } t_j\}.$$

The propositional letters TB_{t_j} are defined by $\mathsf{TB}_{t_j} \equiv (i \in \mathsf{TB}(t_j))$. The reduction in H's length is effectively achieved by *removing* states D_i for those time instants not present in any of the sets TB_{t_j}.

Proposition 7 ([44]). *Let H be a history and Q a range-restricted query. Then (i) $Q(H) = Q^{\mathcal{E}}(\mathcal{E}(H))$ and (ii) for all suffixes H' of H, $\mathcal{E}(H; H')$ can be constructed from $\mathcal{E}(H); H'$ using the same expiration operator.*

Space Bounds Now we need to consider the size of $\mathcal{E}(H)$. Since every single state is bounded by a function of the active data domain size, it is sufficient to concentrate on the number of states kept in $\mathcal{E}(H)$. This number is bounded by the sum of sizes of the $\mathsf{TB}(t_i)$ sets. It is easy to see that the individual sets $\mathsf{TB}_{\mathbf{a}}(t)$ are bounded by the index of the $\sim_Q^H$ relation, which in turn is bounded by a function of the size of the active domain; here we gain up to an exponential factor for every quantifier. Thus $\mathsf{TB}(t_i)$ is bounded by the sum of sizes of $\mathsf{TB}_{\mathbf{a}}(t)$ over all $\mathbf{a}$. As Q is syntactically bounded, the number of different tuples in each subquery can be also bounded by a function of the size of $\mathbf{D}_H$. All together:

$$|\mathcal{E}(H)| \leq \min(f(|\mathbf{D}_H|), |H| + |Q| \cdot |\mathbf{T}_H|)$$

where f may contain up to the quantifier depth of Q nested exponentials.

Example 20 (Single Quantifier Alternation). Consider the query

$$\exists t_1, t_2.t_1 < t_2 \wedge \forall x.P(t_1, x) \iff P(t_2, x).$$

The above query is boolean and can be equivalently expressed as a range-restricted query). It is easy to see that the expired history must keep a copy of every possible subset of the data domain $\mathbf{D}_H$ that appears in the original history to see if there will be a matching set in the future. On the other hand, it is always sufficient to keep just one such copy and remove all others.

The example demonstrates that, in general, we cannot avoid cases where we need to *keep* exponentially many time instants in the size of $\mathbf{D}_H$.

As a final touch needed to make the expired history $\mathcal{E}(H)$ truly independent of the length of H, we need to consider the binary representation of the *indices i* of $D_i \in \mathcal{E}(H)$. It is again easy to see that for range-restricted queries the actual values of the indices i do not matter as long as the relative order is preserved. Therefore we can renumber the states in $\mathcal{E}(H)$ using indices with representation bounded by the size of $\mathbf{D}_H$.

4.3   Beyond Bounded Encodings

While we have been able to define bounded expiration operators for PastTL and 2-FOL queries, in this section we show examples of queries for which bounded encoding cannot exist.

Duplicate Semantics and Aggregation. Consider a closed query over a single unary relational scheme that asks *"have there been more a's than b's in the history?"*, $|\{t : R(t,a)\}| > |\{t : R(t,b)\}|$. for a and b two distinct constants, This query needs to maintain the difference of the number of a's and b's, which in turn requires a logarithmic space in the size of the history; this is a lower bound for any technique and follows from the pigeon-hole principle.

Therefore there is no hope that a technique can maintain queries with aggregation (counting), even when restricted to closed (yes/no) queries, and guarantee that the size of the expired history does not depend on the length of the original history.

Similar argument rules out queries with duplicate semantics: the question asking if there have been more a's than b's in a history can be expressed using using the duplicate-preserving projection and the monus[6] operations.

Retroactive Updates. Similarly to aggregation, allowing even simple retroactive updates makes bounded histories impossible. Consider the following example:

Example 21. Let H be a history containing instances of a single unary relation symbol R. Then we execute the following "transaction":

> **while** $\exists t.R(t,a) \wedge \exists t.R(t,b)$ **do**   { while both a and b exist in R }
>   **delete** $R(t,a)$ **such that** $\forall t'.R(t',a) \supset t' > t$;
>   { delete (chronologically) first a }
>   **delete** $R(t,b)$ **such that** $\forall t'.R(t',b) \supset t' > t$;
>   { delete (chronologically) first b }
> **done**
> **return** $\exists t.R(t,a)$ { return true if R contains an a }

---

[6] The duplicate preserving multiset difference; `EXCEPT ALL` in SQL.

It is easy to see that this transaction returns *true* if and only if there have been more a's than b's in the original history H. Therefore, using the same argument as for counting, there cannot be an equivalent history bounded by a function of the size of the active domain.

Similar transactions can be exhibited for **insert**s and/or **update**s.

Syntactically Bounded Queries. In addition, bounded histories cannot exist for queries returning time instants in their answers (i.e., for queries that are not *syntactically bounded*). Consider the query asking for the last time instant in the history. Then, with the progression of time, the single resulting value is logarithmic in the length of the history ($\log(|\mathbf{T}_H|)$) and therefore not independent of the length of the history.

We conjecture, however, that in these cases the number of items, disregarding the size of their binary encoding, can still be bounded by a function of the size of the active data domain $\mathbf{D}_H$.

Future Temporal Logic with Fixpoints. A more surprising is the case of FutureμTL; the future counterpart of PastμTL. Formally a FutureμTL formula is formula defined by the grammar

$$Q ::= R(\mathbf{x}) \mid F \mid Q \wedge Q \mid \neg Q \mid \exists x.Q \mid \bigcirc Q \mid \mu X.Q.$$

The semantics is defined analogously to the Past fragment; in particular the $\mu X.Q$ *least fixpoint operator* is again interpreted by:

$$H, \theta, i \models \mu X.Q \text{ if } H', \theta, i \models Q$$

where H' extension of H by a "least" interpretation X. The other difference is that the semantics of queries is again defined with respect to the first state of the history.

One would expect that, for finite histories, the future fragment behaves essentially the same as the past one. Unfortunately, that is not the case; consider the query

$$Q = \mu X(x, y).R(x, y) \vee \exists z.\bigcirc R(x, z) \wedge \bigcirc X(z, y)$$

over histories containing instances of a single binary predicate symbol R.

Now assume that there is a bounded expiration operator $\mathcal{E}$ for this query such that $|\mathcal{E}(H)| \leq f(a, q)$ for $f(a, q)$ a function of the sizes of the active data domain and the query, respectively. Let

$$H_n = \langle \underbrace{\emptyset, \emptyset, \ldots, \emptyset}_{n} \rangle$$

be a history with n states, each containing an empty instance of R. Consider the bounded expiration operator $\mathcal{E}$ applied to histories H_n for various values

n. Since the size of the residual history is bounded by a *constant* $f(0, |Q|)$, there must be two sufficiently long histories, H_i and H_j, $i < j$, such that $\mathcal{E}(H_i) = \mathcal{E}(H_j)$. Now consider adding a new state D to both H_i and H_j. The new state D in both cases contains an instance of R that has the form

$$\mathbf{R} = \{(a_0, a_1), (a_1, a_2), \ldots, (a_{i-1}, a_i)\}.$$

It is easy to see that $Q(H_i; D) = \{(a_0, a_i)\}$ while $Q(H_j; D) = \{\}$. However, since $\mathcal{E}(H_i) = \mathcal{E}(H_j)$, we have $\Delta^{\mathcal{E}}(\mathcal{E}(H_i), D) = \Delta^{\mathcal{E}}(\mathcal{E}(H_j), D)$ and thus $Q^{\mathcal{E}}(\mathcal{E}(H_i; D)) = Q^{\mathcal{E}}(H_j; D)$. This violates the requirement of answer preservation for expiration operators; a contradiction. This observation yields the following theorem:

Theorem 2. *There cannot be a bounded expiration operator for queries formulated in FutureμTL.*

This result also shows that there cannot be a bounded expiration operator for fixpoint extensions of 2-FOL, since FutureμTL can be naturally embedded into 2-FOL with fixpoints.

5   Potentially Infinite Histories

So far we only focused on *finite* histories. However, there is an alternative to this approach: the finite histories can be considered to be *finite prefixes* of infinite (or complete) histories. Queries are then evaluated with respect to the infinite histories (using the same semantic definitions as in Section 2.3, the only difference is that an infinite temporal domain is allowed for histories). However, as only finite portion (a prefix) of the history is available at a particular (finite) point in time, we need to define answers to queries with respect to *possible completions* of the prefix to a complete history.

Definition 6. Let H be a finite history, Q a query (in an appropriate query language), and θ a substitution. We say that

- θ is *a potential answer for Q with respect to H* if there is an infinite completion H' of H such that $H', \theta \models Q$.
- θ is *a certain answer for Q with respect to H* if for all infinite completions H' of H we have $H', \theta \models Q$.

The notion of *potential answer* is a direct generalization of the notion of *potential constraint satisfaction* [10].

Unfortunately, the above definition leads to *undecidable satisfaction problems* (for closed formulas in the temporal query languages introduced in Section 2.3) and therefore it is not useful as a basis for query evaluation. In turn, we cannot expect a query driven expiration technique to be applicable in this setting either. The negative results are as follows:

Proposition 8 ([16]). *The satisfaction problem for two-dimensional propositional temporal logic[7] over natural numbers-based time domain is not decidable.*

This proposition rules out temporal relational calculus as it properly contains the above logic. For the weaker query languages based on single-dimensional temporal logics (e.g., FOTL, PastTL, and FutureTL) the results are as follows:

Proposition 9 ([10]). *For past formulas potential constraint satisfaction is undecidable.*

The analysis of FutureTL yields the following:

Proposition 10 ([11]). *For biquantified formulas with no internal quantifiers (called* universal*), potential constraint satisfaction is decidable (in exponential time). For biquantified formulas with a single internal quantifier, potential constraint satisfaction is undecidable.*

While all the above results consider *potential satisfaction* only (and thus for potential query answering), dual results can be obtained for *certain query answering*. In summary, none of the standard temporal query languages can be used in this setting and thus the question of query driven data expiration is a moot point.

6   Related Work

Garbage Collection

Many practical approaches to data expiration [17] are patterned after *garbage collection* algorithms designed for general-purpose programming languages [27]. However, these approaches are limited in their effectiveness (as pointed out in Example 8). This limitation can be traced to the fact that garbage collection algorithms for programming languages are designed to operate in a different setting. In particular:

- Data in typical programming languages is accessed navigationally (i.e., by following pointers). Due to undecidability considerations, the data access is (pessimistically) approximated by *reachability analysis* commonly implemented via mark-and-sweep or generational copying algorithms.
- The starting points for the navigational access are the program variables. Thus, unlike database queries, programming languages do not allow quantification over all instances of a particular data type (objects, records, etc.). Under this restriction the reachability analysis evades the problems of all the data being trivially reachable.

---

[7] In a *two-dimensional temporal logic* the truth of atomic formulas is defined with respect to points a plane rather than a single-dimensional line; temporal connectives in such a logic naturally refer to *two* evaluation coordinates [16].

Temporal and Dynamic Integrity Constraints

A restricted class of queries are the closed formulas (sentences) in a particular query language over histories. Such formulas are often used to enforce integrity constraints over histories. Indeed, the approach to data expiration based on variants of PastTL was originally developed for maintenance of temporal integrity constraints by Chomicki [10] and implemented using triggers on top of an active DBMS [12].

Similarly, the connection between FutureTL sentences and automata theory was utilized to define integrity enforcement mechanisms based on *transition graphs* by Lipeck, Gertz, Saake, et al, [18,23,24,31–34].

Model Checking

The connection between temporal logic (and other modal logics) and automata has been known since the 60s [39]. Linear-time temporal logic (LTL) [15] has been used as a specification language for program properties [36,46]. LTL formulas and finite descriptions of program behavior lead to *model checking*: an approach to the problem of verifying whether a specification obeys a particular LTL property. The two main differences are:

- Model checking uses an *infinite* temporal domain (at least infinite for the future).
- Moreover it assumes complete information about an infinite behavior of the checked system is provided as an input (usually in the form of an automaton).

To preserve decidability (cf. Section 5) the LTL language has to be restricted. In this chapter we traded potential satisfaction for the ability to evaluate queries (by restricting queries to *finite histories*). However, this is not an acceptable solution for Model Checking approaches where potentially infinite computations are essential (e.g., when specifying a communication protocol). Thus, most of the model checking approaches restrict themselves to propositional variants of LTL (or, equivalently, to finite and fixed data domains). This way the logic can be embedded into S1S or S2S yielding decidability [39], and, through the automata connection [43], we can even obtain a PSPACE decision procedure [40].

Materialized View Maintenance

A large body of research [14,19] focuses on maintenance of *materialized views* with respect to their definitions under database updates. However, in this setting, the view definitions are usually restricted to a single database state that can be arbitrarily updated. Yang and Widom [49,50] consider view maintenance for temporal databases. The temporal query language they use is,

however, less expressive than temporal logic and thus their approach is subsumed by the work of Chomicki [10].

An interesting special case is the problem of *materialized view self-maintenance* [25,38]: whether a view can be maintained based purely on its current instance and knowledge of the updates to the base tables. Allowing the use of *auxiliary data*, this problem can be studied as a special case of expiration operator on the history of *database updates* (i.e., the states of the history store the so called delta relations rather than the actual data.)

Interval Encoding of Histories and Constraint Databases

Histories (and temporal databases in general) are often encoded utilizing *interval-based* timestamps [42], a lossless compression technique that takes advantage of the fact that tuples often persist for a contiguous period of time spanning multiple states in the history. In most cases, the *semantics* of the data and queries is still point-based (i.e., temporal quantifiers range over individual time instants rather than over intervals) as pointed out by Chomicki and Toman [13]; the interval encoding can also be viewed as a simple case of *constraint database* [30].

Interval encoding of database histories (or, similarly, transaction-time temporal databases [42]), or any other lossless compression of histories (cf. Example 3), cannot yield a bounded history encoding as pointed out in Section 2.4.

7   Conclusion

This chapter provides a unifying framework in which data expiration techniques can be studied and discusses existing approaches to data expiration within this framework. Particular attention is given to logical (query-driven) expiration operators.

Open Problems

There are many questions left unanswered. We provide a partial list of these:

Complex Temporal Domains.
Through the chapter (with the exception of the discussion of Metric PastTL) we have assumed that the temporal domain is a simple linearly ordered set. That is, the only interpreted relation defined on the temporal domain was the *linear order*. Can the expiration techniques be extended to handling more complex temporal structures?

Space Bounds For Aggregate Queries.
We have shown in Section 4.3 that bounded encoding cannot exist for queries with counting. This, however, only says that the size of the

residual data cannot be bounded by a constant function in the size of the active temporal domain $\mathbf{T}_H$. Can there be a weaker bound, e.g., $|\mathcal{E}(H)| \in O(\log(|\mathbf{T}_H|))$?

Fixpoint Extensions of Future Temporal Logic/2-FOL.

Similarly, the result presented in Theorem 2 shows that a bounded expiration operator for FutureμTL (and consequently for for fixpoint extensions of 2-FOL) cannot exist. But there may still be an expiration operator bounded by a $o(|\mathbf{T}_H|)$ function.

Weaker Query Languages with Computable Potential Answers.

Results in Section 5 seem to disqualify potential query semantics as a candidate for query evaluation. It may be possible to limit temporal queries to languages in which potential satisfaction is decidable. Such restrictions have been studied by Hodkinson, et al, for various first-order extensions of standard temporal and modal logics [8,22]. However, these languages severely restrict first-order quantification (quantification over the data domain) and thus are of little use as general-purpose query languages. Applications of such languages are commonly more appropriate for, e.g., schema languages. This direction has been investigated in the framework of *description logics* and their temporal extensions by Artale, et al, [6]. Another approach would be restricting the power of the first-order language along the lines commonly used in database theory to gain favorable computational properties, e.g., to conjunctive or positive queries [2]. Note however, that computability of potential query semantics does not automatically guarantee the existence of a bounded expiration operator (though, since all these languages can be properly embedded in temporal relational calculus, techniques used in Section 4.2 provide a good starting point). Also, since query emptiness is decidable in these languages, tighter, and possibly optimal, expiration operators may be feasible.

References

1. Serge Abiteboul, Laurent Herr, and Jan Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 49–57, 1996.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

---

3. Foto Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 209–220, 2002.

4. Raajev Alur and Thomas A. Henzinger. Real-Time Logics: Complexity and Expressiveness. *Information and Computation*, 104:35–77, 1993.

5. Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing Memory Requirements for Queries over Continuous Data Streams. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of database systems*, pages 221–232, 2002.

6. Alessandro Artale and Enrico Franconi. A Survey of Temporal Extensions of Description Logics. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 30(1-4), 2001.

7. Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002.

8. Sebastian Bauer, Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. On Non-Local Propositional and Local One-Variable Quantified CTL*. In *International Symposium on Temporal Representation and Reasoning*, pages 2–9. IEEE Press, 2002.

9. Jan Chomicki. Real-Time Integrity Constraints. In *ACM Symposium on Principles of Database Systems*, pages 274–282, 1992.

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

11. Jan Chomicki and Damian Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, 1995.

12. Jan Chomicki and David Toman. Implementing Temporal Integrity Constraints Using an Active DBMS. *IEEE Transactions on Data and Knowledge Engineering*, 7(4):566–582, 1995.

13. Jan Chomicki and David Toman. Temporal Logic in Information Systems. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, pages 31–70. Kluwer, 1998.

14. Latha S. Colby, Timothy Griffin, Leonid Libkin, Inderpal Singh Mumick, and Howard Trickey. Algorithms for Deferred View Maintenance. In *ACM SIGMOD International Conference on Management of Data*, pages 469–480, 1996.

15. E. Allen Emerson. Temporal and Modal Logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 995–1072. Elsevier and MIT Press, 1990.

16. Dov M. Gabbay, Ian M. Hodkinson, and Mark Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*. Oxford University Press, 1994.

17. Hector Garcia-Molina, Wilburt Labio, and Jun Yang. Expiring Data in a Warehouse. In *International Conference on Very Large Data Bases, VLDB'98*, pages 500–511, 1998.

18. Michael Gertz and Udo W. Lipeck. Deriving Optimized Integrity Monitoring Triggers from Dynamic Integrity Constraints. *Data and Knowledge Engineering*, 20(2):163–193, 1996.

19. Ashish Gupta and Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *Data Engineering Bulletin*, 18(2):3–18, 1995.

20. Ashish Gupta and Inderpal Singh Mumick. *Materialized Views: Techniques, Implementations, and Applications*. MIT Press, 1998.

21. Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

22. Ian M. Hodkinson, Frank Wolter, and Michael Zakharyaschev. Monodic Fragments of First-Order Temporal Logics: 2000-2001 A.D. In *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2001, LNCS 2250*, pages 1–23, 2001.

23. Klaus Hülsmann and Gunter Saake. Representation of the Historical Information Necessary for Temporal Integrity Monitoring. In *Advances in Database Technology, EDBT'90*, pages 378–392. Springer-Verlag, LNCS 416, 1990.

24. Klaus Hülsmann and Gunter Saake. Theoretical Foundations of Handling Large Substitution Sets in Temporal Integrity Monitoring. *Acta Informatica*, 28(4), 1991.

25. Nam Huyn. Multiple-View Self-Maintenance in Data Warehousing Environments. In *International Conference on Very Large Data Bases, VLDB'97*, pages 26–35, 1997.

26. Christian S. Jensen. Vacuuming. In Richard T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, pages 447–460. Kluwer Academic Publishers, 1995.

27. Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, 1993. Also available online at `http://www.dina.kvl.dk/~sestoft/pebook /pebook.html`.

28. Ron Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

29. Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 95–104, 1995.

30. Leonid Libkin, Gabriel Kuper, and Jan Paredaens, editors. *Constraint Databases*. Springer, 2000.

31. Udo W. Lipeck. Transformation of Dynamic Integrity Constraints into Transaction Specifications. *Theoretical Computer Science*, 76(1), 1990.

32. Udo W. Lipeck and Dasu Feng. Construction of Deterministic Transition Graphs from Dynamic Integrity Constraints. In *Graph-Theoretic Concepts in Computer Science*, pages 166–179. Springer-Verlag, LNCS 344, 1989.

33. Udo W. Lipeck, Michael Gertz, and Gunter Saake. Transitional Monitoring of Dynamic Integrity Constraints. *IEEE Data Engineering Bulletin*, June 1994.

34. Udo W. Lipeck and Gunter Saake. Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems*, 12(3):255–269, 1987.

35. Udo W. Lipeck and Heren Zhou. Monitoring Dynamic Integrity Constraints on Finite State Sequences and Existence Intervals. In *Workshop on Foundations of Models and Languages for Data and Objects. FMLDO'91*, pages 115–130, 1991.

36. Amir Pnueli. Applications of Temporal Logic to the Specification and Verification of Reactive Systems: a Survey of Current Trends. In *Current Trends in Concurrency*. Springer-Verlag, LNCS 224, 1986.

37. Dallan Quass, Ashish Gupta, Inderpal Singh Mumick, and Jennifer Widom. Making views self-maintainable for data warehousing. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, pages 158–169. IEEE Computer Society, 1996.
38. Dallan Quass, Ashish Gupta, Inderpal Singh Mumick, and Jennifer Widom. Making Views Self-Maintainable for Data Warehousing. In *Fourth International Conference on Parallel and Distributed Information Systems*, pages 158–169, 1996.
39. Michael O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions AMS*, 141:1–35, 1969.
40. A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.
41. Janne Skyt, Christian S. Jensen, and Leo Mark. A foundation for Vacuuming Temporal Databases . *Data and Knowledge Engineering*, 44(1):1–29, 2003.
42. Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.
43. Wolfgang Thomas. Automata on Infinite Objects. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 133–192. Elsevier and MIT Press, 1990.
44. David Toman. Expiration of Historical Databases. In *International Symposium on Temporal Representation and Reasoning*, pages 128–135. IEEE Press, 2001.
45. David Toman and Damian Niwinski. First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *Advances in Database Technology, EDBT'96*, volume 1057, pages 307–324. Springer, 1996.
46. Moshe Vardi and Pierre Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *IEEE Symposium on Logic in Computer Science*, 1986.
47. Moshe Y. Vardi. A Temporal Fixpoint Calculus. In *ACM Symposium on Principles of Programming Languages*, pages 250–259, 1988.
48. Pierre Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1/2):72–99, 1983.
49. Jun Yang and Jennifer Widom. Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. In *Advances in Database Technology, EDBT'98*, pages 389–403, 1998.
50. Jun Yang and Jennifer Widom. Temporal View Self-Maintenance. In *Advances in Database Technology, EDBT'00*, pages 395–412, 2000.

Index