Temporal View Self-Maintenance in a Warehousing Environment*

Jun Yang and Jennifer Widom
Computer Science Department, Stanford University
{junyang,widom}@db.stanford.edu, http://www-db.stanford.edu/

Abstract

Warehouse view *self-maintenance* refers to maintaining materialized views at a data warehouse without accessing source data. Self-maintenance has been studied for nontemporal views, but is even more important when a warehouse stores temporal views over the history of source data, since the source history needed to perform view maintenance may no longer exist. This paper tackles the self-maintenance problem for temporal views. We show how to derive auxiliary data to be stored at the warehouse so that the warehouse views and auxiliary data can be maintained without accessing the sources, and we give algorithms for incremental self-maintenance of all warehouse data.

The temporal view self-maintenance problem is considerably harder than the nontemporal case because a temporal view may need to be maintained not only when source data is modified but also as time advances, and these two dimensions of change interact in subtle ways. We must take both dimensions and their potential interactions into account. We also seek to minimize the amount of auxiliary data required, taking into account different source capabilities and update constraints that are common in temporal warehousing scenarios. While our framework and algorithms are presented using a true temporal data model, our results apply directly to the ad-hoc temporal support (i.e., timestamp attributes in the standard relational model) commonly found in data warehouses today.

1 Introduction

A *data warehouse* is a repository of data that has been extracted and integrated from multiple operational *data sources* for efficient querying and analysis. Many warehouse applications are *temporal* in nature, i.e., they are based not only on the up-to-date source information, but also on the history of the data at the sources. On the other hand, operational sources usually do not store their entire history. While some sources may store partial history, many *nontemporal* sources store only the current state of their data. For example, a company's sales department database might keep track of the current pending orders, and remove orders once they are filled. Nevertheless, the company analysts might want to pose queries over all orders that have ever been submitted.

Data warehouses typically materialize *views* to support the expected queries of warehouse users and applications. In [YW98a] we presented a temporal data warehousing framework in which the warehouse materializes and incrementally maintains temporal views over the history of source data. Since sources may be nontemporal, we cannot expect to be able to query them for the source data history that may be needed to maintain temporal warehouse views. The solution is to make the

*This work was supported by DARPA and the Air Force Rome Laboratories under contracts F30602-95-C-0119 and F30602-96-1-0312, and by the Advanced Research and Development Committee of the Community Management Staff as a project in the MDDS Program.

views *self-maintainable* by storing *auxiliary data* at the warehouse so that the warehouse data as a whole can be maintained without accessing any source data. The notion of materialized view self-maintenance was introduced for nontemporal views in [BCL89, GJM96, QGMW96]. In this paper, we tackle the self-maintenance problem for temporal views, including in our view definition language a class of commonly used temporal aggregates termed *moving-window aggregates* [NA89]. Note that while self-maintenance for nontemporal views may largely be a matter of convenience or efficiency [QGMW96], for temporal views over nontemporal sources self-maintenance usually is required.

The temporal version of the view self-maintenance problem poses significant challenges not found in the nontemporal version. Temporal views may be affected not only by data updates, but also as time advances. For example, in the sales department database, an order that was placed yesterday may not be considered "old" now, but will turn old if it remains outstanding after a month. To make temporal views self-maintainable, we must consider both dimensions of change (data and time), and more importantly, their potential interactions. As we shall see, the auxiliary data required for handling both dimensions of change is not simply the union of the data required for handling the two dimensions independently.

An obvious brute-force solution is to store the entire history of all source data at the warehouse. Because history grows monotonically, this solution eventually fails when the warehouse runs out of storage space. Even if storage is not a concern, it is helpful to identify the historical data not required for view maintenance and move it off-line in order to speed up regular operations. In this paper, we seek to minimize the amount of auxiliary data needed for temporal view self-maintenance, taking into account different source capabilities and update constraints that are common in warehousing scenarios. We consider not only nontemporal sources, but also more powerful sources that represent time and allow retroactive updates, possibly within a fixed-length *update window*. For example, the payroll department of a company may maintain a database that records the days that each employee has worked. It may be possible to modify the time records within last seven days, but once records become more than a week old they are considered permanent. We shall address how the length of the update window affects the amount of auxiliary data needed for self-maintenance. Moreover, note that in this example each employee tuple (indicating time at work) may be "alive" for multiple disjoint time periods. In our earlier sales department example, each order tuple is "alive" for only one time period. Whether a source tuple can be "alive" for a set of periods versus a single period also has a dramatic impact on the amount of auxiliary data needed.

We base our work on a true temporal data model and a temporal query language equivalent to a subset of TSQL2 [BJS95]. However, we must emphasize that our results are very applicable to existing warehouses and operational sources with ad-hoc temporal support. These systems typically store temporal information as normal attributes in the relational model, and query it using SQL. An ad-hoc temporal warehouse usually materializes the entire history of all source data, and leaves it to the warehouse administrator to control the purging of historical data [GMLY98]. By solving the self-maintenance problem, our work provides a way to automatically identify and remove warehouse data no longer needed for view maintenance. To apply our results in a warehouse with ad-hoc temporal support, the warehouse designer can start with our data model and view definition language, and then map them to the actual ones implemented by the warehouse. This process usually is not difficult, and takes place only during the warehouse design phase.

The paper proceeds as follows. Section 2 provides basic definitions, the problem setting, and several examples to illustrate the intuition behind temporal view self-maintenance. Section 3 shows how to make a temporal selection view self-maintainable, which forms the core result of this paper. Section 4 extends the result to handle temporal joins. Section 5 covers temporal views that are *snapshot-reducible* [Sno87] and may include projection, difference, and union operators. Section 6 deals with temporal aggregates. Section 7 addresses additional issues and special cases. Finally, Section 8 discusses related and future work.

2 Preliminaries

2.1 Data Model and View Definition Language

Our temporal data model is essentially *BCDM* (*Bitemporal Conceptual Data Model* [JSS94]) restricted to one time dimension. The time domain $\mathbb{T}$ is a set of *time instants*, or *chronons*, isomorphic to the natural numbers. A temporal relation schema has the form $(A_1, A_2, ..., A_m, \mathtt{T})$, where the A_i's are *explicit attributes* and $\mathtt{T}$ is the implicit *time attribute*. Explicit attributes form a key, i.e., $A_1, A_2, ..., A_m \rightarrow \mathtt{T}$. Values for explicit attributes come from regular value domains. The value for the time attribute is a nonempty set of time instants drawn from $\mathbb{T}$. The time attribute also can be viewed as a set of maximal, nonoverlapping time periods, each encoded by a pair of start and end time instants.

Intuitively, a temporal relation $R(A_1, A_2, ..., A_m, \mathtt{T})$ records the history of a nontemporal relation $R^{nt}(A_1, A_2, ..., A_m)$. A temporal tuple $r \in R$ is interpreted as follows: the nontemporal tuple $r^{nt} = \langle r.A_1, r.A_2, ..., r.A_m \rangle$ is present in the state of R^{nt} at time instant t iff $t \in r.\mathtt{T}$. In other words, the time attribute records the times during which the tuple is "alive". We require that $r.\mathtt{T}$ does not extend beyond t_{now}, the current time. However, the last time period of $r.\mathtt{T}$ may end with a special symbol $\mathtt{NOW}$, meaning that r^{nt} is alive in the current snapshot of R^{nt}, and will continue to stay alive as time advances. A detailed discussion of our treatment of $\mathtt{NOW}$ can be found in [YW98a], and a more general discussion of $\mathtt{NOW}$ in temporal databases appears in [CDI$^+$97].

We now define the five operators in our basic temporal relational algebra. The aggregate operator will be introduced separately in Section 6. In the following, R, R_i, S are temporal relations, r, r_i, s are tuples from the respective relations, and A_R, A_{R_i}, A_S denote the sets of explicit attributes in their schemas. Let $r.A$ denote the part of r containing values for the attributes in a set A. To facilitate definition, we define a helper function $h(R, r', A) \stackrel{\text{def}}{=} \bigcup_{(r \in R \ \wedge \ r.A = r'.A)} r.\mathtt{T}$, which returns the union of the time attributes of all tuples in R that agree with tuple r' on the attributes in A.

- **(Selection)** $\quad \sigma_p(R) \stackrel{\text{def}}{=} \{r \mid r \in R \ \wedge \ p(r)\}$
- **(Join)** $\quad \bowtie_p(R_1, ..., R_n) \stackrel{\text{def}}{=} \{\langle r_1.A_{R_1}, ..., r_n.A_{R_n}, c\rangle \mid r_1 \in R_1 \ \wedge ... \wedge \ r_n \in R_n \ \wedge \ p(r_1, ..., r_n) \ \wedge \ c = r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \ \wedge \ c \neq \varnothing\}$
- **(Projection)** $\quad \pi_A(R) \stackrel{\text{def}}{=} \{\langle r.A, c\rangle \mid r \in R \ \wedge \ c = h(R, r, A)\}$
- **(Difference)** $\quad R - S \stackrel{\text{def}}{=} \{\langle r.A_R, c\rangle \mid r \in R \ \wedge \ c = r.\mathtt{T} - h(S, r, A_R) \ \wedge \ c \neq \varnothing\}$
- **(Union)** $\quad R \cup S \stackrel{\text{def}}{=} \{\langle u, c\rangle \mid (\exists r \in R : u = r.A_R \ \wedge \ c = r.\mathtt{T} \cup h(S, r, A_R)) \ \vee \ (\exists s \in S : u = s.A_R \ \wedge \ c = s.\mathtt{T} \cup h(R, s, A_R))\}$

The predicate p used in the temporal selection and join operators may be a standard nontemporal predicate referencing explicit attributes, a temporal predicate referencing the time attribute

T and/or the special symbol NOW, or a combination of both, connected via Boolean operators $\wedge$, $\vee$, and $\neg$. The exact list of temporal predicates that we study in this paper appears in Appendix A, although other predicates can be handled by our framework as well. Here, we briefly describe some of the atomic temporal predicates we consider. TS represents the start time of the earliest period in T; TE represents the end time of the last period in T. TS and TE can be compared to any time instant $t \in \mathbb{T}$ using the operators $=$, $>$, $<$, etc. They also can be compared to a NOW-relative time such as $\mathtt{NOW} - \mathtt{3\ weeks}$. During query evaluation, temporal predicates are evaluated with the implicit binding $\mathtt{NOW} = t_{now}$, where t_{now} is the current time. Predicate (T overlaps $[t_s, t_e]$) tests whether the time attribute overlaps the period $[t_s, t_e]$; predicate (T contains $[t_s, t_e]$) tests whether T contains $[t_s, t_e]$. Again, t_s and t_e may be NOW-relative. Finally, length(T) returns the sum of the lengths of the periods in T, which can be compared to any constant length (such as 1 year).

2.2 Temporal Data Warehouse

Each data source has a number of *source relations* monitored and exported by an *extractor* [Wid95]. Conceptually, for each source relation, the temporal warehouse has a corresponding *temporal base relation* representing the source relation's history. In this paper, we assume that our temporal view is defined over the conceptual temporal base relations, and that we are provided with updates to these relations using, e.g., the techniques in [YW98a]. We do not assume or require that the temporal base relations actually exist. The results in this paper show how to modify the temporal view definition and/or materialize additional views in order to make the temporal warehouse fully self-maintainable.

Maintenance of temporal views involves two procedures: *view refresh* and *change propagation* [YW98a]. View refresh is the procedure for updating temporal views with respect to time advances. Since it is generally inefficient to update views at every clock tick, view refresh usually is triggered either implicitly by a request to access the view contents, or explicitly by a request from the warehouse administrator. Change propagation is the procedure for updating views with respect to data changes. Source updates are detected by the extractor and converted to updates on the corresponding temporal base relations, which then trigger change propagation at the warehouse. To make a temporal warehouse self-maintainable, we must ensure that the warehouse contains all data needed for both view refresh and change propagation.

Sources with temporal support often allow retroactive updates within some *update window*. Formally, an update window is a NOW-relative time period of the form $[\mathtt{NOW} - l, \mathtt{NOW}]$, where $l \geq 0$. An update μ on a tuple r with update window $[\mathtt{NOW} - l, \mathtt{NOW}]$ may modify the value of $r.\mathtt{T}$ in any arbitrary way within $[\mathtt{NOW} - l, \mathtt{NOW}]$. However, the part of $\mu(r).\mathtt{T}$ before $\mathtt{NOW} - l$ must remain the same as the original $r.\mathtt{T}$. As an important special case, a nontemporal source has an update window of $[\mathtt{NOW}, \mathtt{NOW}]$, i.e., only the current state of the data can be updated. An update of explicit attributes becomes two updates of time attributes, as will be seen in Example 2.4, so we need not consider updates of explicit attributes separately. In practice, updates to a relation R are processed in batches $\triangledown R$ and $\triangle R$, which together describe the net effect of the updates as $R \leftarrow R - \triangledown R \cup \triangle R$.

Besides the update-window constraint, we also consider the *single-period constraint* which is common in warehousing scenarios. This constraint requires the time attribute of a source history tuple to be a single time period rather than a set of time periods. For instance, the single-period constraint holds for the order history in the sales department example given earlier, but does not

	OrdID	T
r_1	1	$[09/01, 09/11]$
r_2	2	$[09/22, 09/27]$
r_3	3	$[09/23, 09/28]$
r_4	4	$[09/24, 09/29]$
r_5	5	$[09/25, \texttt{NOW}]$

Figure 1: Contents of R on 10/01.

	Emp	Dept	T
s_1	A	marketing	$[09/14, 09/18] \cup [09/21, 09/25]$
s_2	A	sales	$[09/28, \texttt{NOW}]$
s_3	B	sales	$[09/23, 09/25] \cup [09/28, \texttt{NOW}]$
s_4	C	payroll	$[09/28, \texttt{NOW}]$
s_5	D	sales	$[09/14, 09/18] \cup [09/28, \texttt{NOW}]$

Figure 2: Contents of S on 10/01.

hold for the employee work history in the payroll department example.

2.3 Examples

In this section, we provide examples that yield insight into the temporal view self-maintenance problem. Examples 2.1, 2.2, and 2.3 illustrate how various constraints affect the amount of auxiliary data needed. Example 2.4 considers a temporal selection view with a composite predicate. We intentionally provide several examples that are simple and easy to follow, but the general case can be quite complicated and subtle, as we shall see in Example 2.5.

Example 2.1 (Update window $[\texttt{NOW}, \texttt{NOW}]$, single-period constraint) Consider a conceptual temporal base relation that records the order history for a sales department. A simplified version of this relation, $R(\texttt{OrdID}, \texttt{T})$, is shown in Figure 1. The granularity of the time domain is one day, and the current time is 10/01. Each order tuple is "alive" when the order is pending. We assume that orders are pending for only one time period. For example, $r_1.\texttt{T} = [09/01, 09/11]$ means that order 1 was submitted on 09/01 and filled on 09/12.

Suppose the customer service department has a policy of calling customers two days after their order has been filled to make sure they have received the shipment. For this purpose, we define a temporal warehouse view $V \stackrel{\text{def}}{=} \sigma_{\texttt{TE=NOW-3 days}}(R)$. Currently on 10/01, only r_3 is in the view. However, data in addition to V itself may need to be stored at the warehouse if we want to correctly maintain V when time advances and/or R is updated, without relying on historical data retrieved from the source.

Consider time advances first. Say that one day has passed, and NOW has changed from 10/01 to 10/02. As a result, r_4 will satisfy $(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ and should be inserted into V. To be able to perform this insertion on 10/02, we must store r_4 in advance.

Now, let us also take source updates into account. Assume that the sales department database only allows updates at the current time, i.e., its update window is $[\texttt{NOW}, \texttt{NOW}]$. Suppose on 10/01, order 5 gets filled and is removed from the sales department database. This source deletion translates into an update on the temporal base relation R as $\triangledown R = \{\langle 5, [10/01, \texttt{NOW}]\rangle\}$, which changes r_5 to $r_5' = \langle 5, [09/25, 09/30]\rangle$. (See [YW98a] for details on how this translation is performed.) Then, when time advances to 10/03, r_5' will be in V. In order to compute r_5', we need to store r_5 in advance, because $r_5'.\texttt{T}$ cannot be derived from $\triangledown R$ alone.

To summarize, to make V self-maintainable, we must store not only r_3, which is currently in V, but also r_4 and r_5, which could potentially contribute to V after going through some sequence of time advances and/or data updates. On the other hand, we need not store r_1 or r_2. On 10/01,

r_1.TE and r_2.TE are already less than $\texttt{NOW} - \texttt{3 days}$. Since NOW always grows bigger as time advances, ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$) will never become true unless T is updated. Both r_1.T and r_2.T have terminated far before the start of the update window, and new time instants cannot be added to r_1.T or r_2.T because that would create disjoint time periods, violating the single-period constraint.

In this example, the fact that we need to save r_4 and r_5 but not r_1 or r_2 is intuitively quite obvious. However, the reasoning involved is illustrative of the techniques underlying our results, which apply to much more complex views. □

Example 2.2 (Update window [$\texttt{NOW} - 7\,\texttt{days}, \texttt{NOW}$], single-period constraint) The setup is identical to Example 2.1, except now we assume that the source has an update window of [$\texttt{NOW} - \texttt{7 days}, \texttt{NOW}$], which therefore includes r_2.TE. Thus, in addition to storing r_3, r_4, and r_5, we must store r_2, because it is possible for a source update on 10/01 to retroactively change r_2.T to [09/22, 09/28], which satisfies ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$). We need not store r_1 since r_1.TE lies well before the update window and the single-period constraint still holds. □

Example 2.3 (Update window [$\texttt{NOW} - 7\,\texttt{days}, \texttt{NOW}$], no single-period constraint) Continuing with Example 2.2, suppose that we also drop the single-period constraint. Now, r_1 must be stored as well: we could update the value of r_1.T inside the current update window to [09/27, 09/28], resulting in $r_1.\texttt{T} = [\texttt{09/01}, \texttt{09/11}] \cup [\texttt{09/27}, \texttt{09/28}]$ which satisfies ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$). □

Example 2.4 (Composite selection predicate) Consider a conceptual temporal base relation $S(\texttt{Emp}, \texttt{Dept}, \texttt{T})$, which records the set of days worked by a given employee in a given department. The corresponding source relation is stored in the payroll department database, with an update window of [$\texttt{NOW} - \texttt{7 days}, \texttt{NOW}$]. Figure 2 shows the current contents of S on 10/01.

Suppose the manager of the sales department wants to keep an eye on the inexperienced employees, i.e., those who have worked in the department for fewer than five days altogether. Accordingly, we define a temporal warehouse view $U \stackrel{\text{def}}{=} \sigma_{\texttt{Dept}=\texttt{sales} \,\wedge\, \texttt{length(T)} < \texttt{5 days}}(S)$.

First, note that in order to maintain U, we need not store any S tuples that do not already satisfy ($\texttt{Dept} = \texttt{sales}$). The truth value of this predicate is not affected by time advances or updates of T. As for updates of explicit attributes, they are naturally modeled as pairs of updates of T. For example, if employee A moves from the sales department back to the marketing department on 10/01, this update of Dept will translate into $\triangledown R = \{\langle \texttt{A}, \texttt{sales}, [\texttt{10/01}, \texttt{NOW}]\rangle\}$ and $\triangle R = \{\langle \texttt{A}, \texttt{marketing}, [\texttt{10/01}, \texttt{NOW}]\rangle\}$. (Again, see [YW98a] for details.) $\triangledown R$ will update s_2.T to [09/28, 09/30], while $\triangle R$ will update s_1.T to $[\texttt{09/14}, \texttt{09/18}] \cup [\texttt{09/21}, \texttt{09/25}] \cup [\texttt{10/01}, \texttt{NOW}]$. Neither $\triangledown R$ nor $\triangle R$ will change the truth value of ($\texttt{Dept} = \texttt{sales}$) for any tuple. Therefore, we need not store s_1 or s_4 because these tuples can never satisfy ($\texttt{Dept} = \texttt{sales}$) in the future.

Among the remaining tuples, s_2 is already in U. To make U self-maintainable, we must also store s_3, but not s_5. The reason is as follows. Although the current value of s_3.T indicates that employee B already has worked in the sales department for seven days, it is possible that on 10/01 the payroll department realizes that there is a mistake in the records: employee B has in fact been on leave for training since 09/28. Thus, a correction is made, and s_3.T is updated to [09/23, 09/25], which now satisfies ($\texttt{length(T)} < \texttt{5 days}$). On the other hand, for employee D, the part of s_5.T outside the update window is already five days long. No matter how we modify the value of s_5.T within the update window, the length of s_5.T will still be at least five days. Moreover, the length of a time attribute will never decrease as time advances. Therefore, we need not store s_5.

Again, the end result of our reasoning should be fairly obvious, but the reasoning itself generalizes to much more complex views, as we will see in Section 3. □

Example 2.5 (Seemingly simple selection) Consider a view $W \stackrel{\text{def}}{=} \sigma_{\texttt{TE}=t}(R)$, where R is the same as in Example 2.2, and t is some fixed time instant, say, `12/25`. It turns out that in order to make W self-maintainable, we need to store $\sigma_q(R)$, where q is:

$$(\texttt{TE} = t) \vee \big((\texttt{NOW} - \texttt{8 days} = t) \wedge (\texttt{TE} \geq t) \wedge (\texttt{TS} \leq t)\big) \vee \big((\texttt{NOW} - \texttt{7 days} \leq t) \wedge (\texttt{TE} \geq \texttt{NOW} - \texttt{8 days})\big)$$

An interesting property of q is that it depends on the distance between t and the current time, which changes as the current time advances.

Given the definition of q as provided above, it is not terribly difficult to verify that $\sigma_q(R)$ indeed makes W self-maintainable. However, a number of subtleties exist, and it should be evident to the reader that coming up with a proper q is not a trivial process. How do we find q automatically for arbitrary selection predicates? How do we guarantee that $\sigma_q(R)$ is self-maintainable? How do we know whether $\sigma_q(R)$ contains the minimum amount of information required? The rest of this paper provides a systematic approach to answering these questions, as well as techniques for handling other types of temporal views including joins and aggregates. □

3 Temporal Selection Views

As we have seen in the examples, even a selection view may not be self-maintainable without storing auxiliary information. To make a temporal selection view $\sigma_p(R)$ self-maintainable, we derive a "superview" $\sigma_{\alpha(p)}(R)$ by relaxing the selection predicate from p to $\alpha(p)$. Instead of the original view, we materialize and maintain the superview. The superview must contain not only the current contents of the original view, but also all tuples with the potential of contributing to the original view after going through a sequence of time advances and/or data updates. Furthermore, the superview must itself be self-maintainable. In this section, we study how to find the smallest superview that meets these requirements.

As mentioned in Section 2.2, contents of a temporal view are affected by two types of changes: time advances and data updates. Since these two types of changes can interact in rather intricate ways, we first consider them separately. We show how to derive $\alpha^t(p)$ such that $\sigma_{\alpha^t(p)}(R)$ is self-maintainable with respect to time advances, and $\alpha^\mu(p)$ such that $\sigma_{\alpha^\mu(p)}(R)$ is self-maintainable with respect to data updates. Then, we show how to combine $\alpha^t(p)$ and $\alpha^\mu(p)$ together to derive $\alpha(p)$ such that $\sigma_{\alpha(p)}(R)$ is self-maintainable with respect to both time advances and data updates. As we will see, simply taking the disjunction of $\alpha^t(p)$ and $\alpha^\mu(p)$ is not sufficient.

Before delving into the details we must introduce some additional notation. Recall that temporal predicates are always evaluated with a binding for `NOW`. We use $[p]_t(r)$ to denote that predicate p is evaluated on tuple r with the binding $\texttt{NOW} = t$. If the $[\cdot]_t$ notation is omitted, p is evaluated with the default binding $\texttt{NOW} = t_{now}$, where t_{now} is the current time. Similarly, we use $[\mu]_t(r)$ to denote that update μ is applied to r at time t.

3.1 Handling Time Advances: α^t

In this subsection we ignore data updates and concentrate on the problem of making a temporal selection view self-maintainable with respect to time advances alone. At the very least, the superview must contain enough information to maintain the original view as time advances. Therefore, we must relax the selection predicate to also select all tuples that could satisfy the original predicate at a future time, leading to the following definition of $\bar{\alpha}^t$ as our first guess for α^t.

Definition 3.1 **($\bar{\alpha}^t$)** Let t_{now} be the current time and p a selection predicate. Define $\bar{\alpha}^t(p)$ as a selection predicate over a temporal tuple r: $(\bar{\alpha}^t(p))(r) \stackrel{\text{def}}{=} \exists t \geq t_{now} : [p]_t(r)$. □

Intuitively, $\bar{\alpha}^t(p)$ selects all and only those tuples that either satisfy p now or could satisfy p in the future if there are no data updates. Any correct superview of $\sigma_p(R)$ must necessarily contain all these tuples because they are either in $\sigma_p(R)$ already or might be added to $\sigma_p(R)$ as time advances. In effect, $\bar{\alpha}^t$ provides a lower bound for α^t, but we still need to ensure the self-maintainability of the superview. With $\bar{\alpha}^t$ defined, we can now be more precise about what constitutes a "correct" definition of α^t.

Definition 3.2 **(Correct α^t)** Let p be a selection predicate. A definition of α^t is *correct* if the following properties hold: (i) $\alpha^t(p) \leftarrow p$; (ii) $\alpha^t(p) \leftarrow \bar{\alpha}^t(p)$; (iii) $\alpha^t(p) \leftarrow \bar{\alpha}^t(\alpha^t(p))$. □

Property (i) ensures that the superview contains the original view. Property (ii) ensures that the superview contains enough information to maintain the original view with respect to time advances. Property (iii) ensures that the superview also contains enough information to maintain itself with respect to time advances, i.e., the superview is self-maintainable with respect to time advances. Strictly speaking, either (i) or (ii) is redundant: (i) follows from (ii), and (ii) follows from (i) and (iii). Nevertheless, it is instructive to list all three properties because each one translates into a different intuitive requirement.

A trivially correct definition of α^t simply sets $\alpha^t(p)$ to `true` for any p, which corresponds to the simple solution of always materializing complete temporal base relations. Fortunately, we can do better. By Def. 3.1, $\bar{\alpha}^t$ already satisfies Properties (i) and (ii) of Def. 3.2. We still need to verify that Property (iii) holds; that is, $\bar{\alpha}^t(p)$ should select not only all tuples with the potential of satisfying p, but also those with the potential of satisfying $\bar{\alpha}^t(p)$ itself in the future. It turns out that $\bar{\alpha}^t$ is idempotent, i.e., $\bar{\alpha}^t(p) \leftrightarrow \bar{\alpha}^t(\bar{\alpha}^t(p))$. Therefore, $\bar{\alpha}^t$ is a correct definition of α^t; in fact, it is the best α^t that we can hope for, as shown by the following theorem.

Theorem 3.1 **(Correctness & minimality of $\bar{\alpha}^t$)** $\bar{\alpha}^t$ is a correct definition of α^t. Furthermore, it is minimal: given any correct definition of α^t, $\bar{\alpha}^t(p) \rightarrow \alpha^t(p)$ for any selection predicate p. □

Theoretically, then, $\bar{\alpha}^t$ is an ideal definition of α^t. Computationally, however, it is problematic. Def. 3.1 cannot be evaluated by typical query engines because it contains an existentially quantified variable t over an infinite domain $\mathbb{T}$. Therefore, we will devise $\hat{\alpha}^t$ as a computationally feasible alternative to $\bar{\alpha}^t$. To compute $\hat{\alpha}^t(p)$ for a composite predicate p, we first apply DeMorgan's Laws to push all negations in p down to the level of atomic predicates. Then, for each atomic or negated atomic predicate q in p, we replace q with a quantifier-free predicate logically equivalent to $\bar{\alpha}^t(q)$ that can be evaluated by typical query engines. In Appendix A, we have specified $\hat{\alpha}^t$ for all atomic and negated atomic predicates in our temporal predicate language. Table 1 covers the general case where `T` may be a set of time periods. Table 2 assumes the single-period constraint. Computing $\hat{\alpha}^t$ thus involves nothing more than pushing all negations down and performing lookups in the appropriate table. The formal definition of $\hat{\alpha}^t$ is given below, followed by a theorem stating its correctness and an example showing its computation.

Definition 3.3 ($\hat{\alpha}^t$) Let p be a selection predicate with negations pushed down to the level of atomic predicates. We define $\hat{\alpha}^t(p)$ inductively on the structure of p. If p is an atomic or negated atomic predicate, $\hat{\alpha}^t(p)$ is defined as a quantifier-free predicate logically equivalent to $\bar{\alpha}^t(p)$ (specified in Tables 1 and 2 of Appendix A). $\hat{\alpha}^t(p_1 \vee p_2)$ is defined as $\hat{\alpha}^t(p_1) \vee \hat{\alpha}^t(p_2)$. $\hat{\alpha}^t(p_1 \wedge p_2)$ is defined as $\hat{\alpha}^t(p_1) \wedge \hat{\alpha}^t(p_2)$. □

Theorem 3.2 (Correctness of $\hat{\alpha}^t$) $\hat{\alpha}^t$ is a correct definition of α^t according to Def. 3.2. □

Example 3.1 Recall Example 2.4. We demonstrate how to compute $\hat{\alpha}^t(p_1 \wedge p_2)$, where p_1 is ($\texttt{Dept} = \texttt{sales}$) and p_2 is ($\texttt{length(T)} < \texttt{5 days}$). Here T may be a set of time periods. According to Rows 1 and 23 of Table 1, $\hat{\alpha}^t(p_1)$ is ($\texttt{Dept} = \texttt{sales}$), while $\hat{\alpha}^t(p_2)$ is ($\texttt{length(T)} < \texttt{5 days}$). Therefore, $\hat{\alpha}^t(p_1 \wedge p_2)$ is ($\texttt{Dept} = \texttt{sales}$) $\wedge$ ($\texttt{length(T)} < \texttt{5 days}$), which happens to be $p_1 \wedge p_2$.

Intuitively, if a tuple does not satisfy $p_1 \wedge p_2$ now, it cannot possibly satisfy $p_1 \wedge p_2$ in the future without data updates. For ($\texttt{Dept} = \texttt{sales}$), time advances have no effect on the truth value of a nontemporal predicate; for ($\texttt{length(T)} < \texttt{5 days}$), length(T) never decreases as time advances. □

3.1.1 Minimality of $\hat{\alpha}^t$

By Def. 3.3, $\hat{\alpha}^t$ is equivalent to $\bar{\alpha}^t$ for all atomic predicates and their negations. Therefore, according to Thm. 3.1, $\hat{\alpha}^t$ is also the minimal definition of α^t for atomic predicates. For composite predicates, Def. 3.3 preserves the minimality of $\hat{\alpha}^t$ over disjunction. Intuitively, $p_1 \vee p_2$ will be true in the future iff either p_1 or p_2 will be true in the future. Formally, $\bar{\alpha}^t(p_1 \vee p_2) \leftrightarrow \bar{\alpha}^t(p_1) \vee \bar{\alpha}^t(p_2)$, so if $\hat{\alpha}^t$ and $\bar{\alpha}^t$ are equivalent for both p_1 and p_2, they should also be equivalent for $p_1 \vee p_2$. On the other hand, Def. 3.3 does not always preserve minimality over conjunction. Intuitively, if $p_1 \wedge p_2$ will be true in the future, then both p_1 and p_2 will be true in the future. However, the converse does not hold because p_1 and p_2 may not become true at the same time. Formally, $\bar{\alpha}^t(p_1 \wedge p_2) \rightarrow \bar{\alpha}^t(p_1) \wedge \bar{\alpha}^t(p_2)$, but the converse may or may not be true. Example 3.1 happened to be one where minimality is preserved over conjunction. The following example illustrates the loss of minimality over conjunction.

Example 3.2 Consider a predicate $p_1 \wedge p_2$, where p_1 is ($\texttt{NOW} > \texttt{10/02}$) and p_2 is ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$). According to Row 3 of Table 1 (or Table 2; the single-period constraint happens to have no effect on this example), $\hat{\alpha}^t(p_1)$ is true, since NOW will eventually move past 10/02 as time advances. According to Row 16, $\hat{\alpha}^t(p_2)$ is ($\texttt{TE} \geq \texttt{NOW} - \texttt{3 days}$) $\wedge$ (TE is not NOW). Intuitively, if TE is some fixed time instant on or after the current value of NOW − 3 days, p_2 will eventually become true when NOW − 3 days catches up with TE in time. On the contrary, if TE is before NOW − 3 days, p_2 will never become true since NOW − 3 days only grows bigger as time advances. Furthermore, a tuple whose TE is NOW can never satisfy p_2 because NOW always comes after NOW − 3 days. Hence, both $\hat{\alpha}^t(p_1)$ and $\hat{\alpha}^t(p_2)$ are minimal, as expected.

By Def. 3.3, $\hat{\alpha}^t(p_1 \wedge p_2)$ is $\hat{\alpha}^t(p_1) \wedge \hat{\alpha}^t(p_2)$, which reduces to ($\texttt{TE} \geq \texttt{NOW} - \texttt{3 days}$) $\wedge$ (TE is not NOW). Unfortunately, we have relaxed the original predicate too much. To see why, suppose that the current time is 10/01, and there is a tuple whose TE = 09/29. This tuple currently satisfies $\hat{\alpha}^t(p_1 \wedge p_2)$, but it will never satisfy $p_1 \wedge p_2$. The reason is that p_1 will be true only after 10/02, while p_2 will be true only on 10/02. □

To alleviate the loss of minimality, it is possible to refine $\hat{\alpha}^t$ by first augmenting the conjunction with additional predicates that logically follow from the conjunction, and then applying $\hat{\alpha}^t$ to the augmented conjunction. For instance, we can augment $p_1 \wedge p_2$ in Example 3.2 with $(\texttt{TE} > \texttt{09/29})$, which follows from $p_1 \wedge p_2$. Applying $\hat{\alpha}^t$ to the augmented conjunction $p_1 \wedge p_2 \wedge (\texttt{TE} > \texttt{09/29})$ and simplifying the result, we get $(\texttt{TE} \geq \texttt{NOW} - \texttt{3 days}) \wedge (\texttt{TE is not NOW}) \wedge (\texttt{TE} > \texttt{09/29})$. This predicate is indeed a minimal α^t for $p_1 \wedge p_2$ and will correctly rule out the tuple whose $\texttt{TE} = \texttt{09/29}$.

To automatically infer predicates that follow from a given conjunction a temporal theorem prover would be needed. However, without such a theorem prover our framework is still correct: although $\hat{\alpha}^t$ may not be minimal for certain conjunctive predicates, $\hat{\alpha}^t$ is still a correct definition of α^t. The practical implication of a nonminimal $\hat{\alpha}^t$ is that some tuples may be saved unnecessarily. However, as seen above, the scenarios affected are relatively unusual.

3.2 Handling Data Updates: α^μ

We now turn to the problem of making a temporal selection view self-maintainable with respect to data updates alone. Similar to handling time advances, for a view $\sigma_p(R)$ our approach is to relax the selection predicate to $\alpha^\mu(p)$ such that $\sigma_{\alpha^\mu(p)}(R)$ is self-maintainable with respect to data updates. As a first step, we define $\bar{\alpha}^\mu(p)$ to select all tuples that either satisfy p now or may satisfy p after an update. Recall that we need only consider updates to time attributes (Section 2.2).

Definition 3.4 ($\bar{\alpha}^\mu$) Let t_{now} be the current time and p a selection predicate. Suppose that the update window is $[\texttt{NOW} - l, \texttt{NOW}]$. Define $\bar{\alpha}^\mu(p)$ as a selection predicate over a temporal tuple r: $(\bar{\alpha}^\mu(p))(r) \stackrel{\text{def}}{=} \exists \mu \text{ in } [t_{now} - l, t_{now}] : p(\mu(r))$. Here, μ is an update of $\texttt{T}$ that may possibly leave $\texttt{T}$ unchanged but may not change $\texttt{T}$ to $\varnothing$. □

Intuitively, $\sigma_{\bar{\alpha}^\mu(p)}(R)$ contains the contents of $\sigma_p(R)$ plus enough auxiliary information to maintain $\sigma_p(R)$ with respect to data updates, and nothing more. When an update arrives, we check whether it is for a tuple stored in $\sigma_{\bar{\alpha}^\mu(p)}(R)$. If so, we have enough information to apply the update and maintain the view. If not, we know that this update cannot cause any tuple to satisfy p, because all tuples with the potential of satisfying p after an update are stored in $\sigma_{\bar{\alpha}^\mu(p)}(R)$.

A nice duality exists between α^μ for data updates and α^t for time advances. The following definitions and theorems are identical in form to those in Section 3.1. We first formalize the notion of a correct α^μ using $\bar{\alpha}^\mu$, and show that $\bar{\alpha}^\mu$ is a correct definition of α^μ and also the minimal one. We then define $\hat{\alpha}^\mu$ as a computable alternative to $\bar{\alpha}^\mu$, in the same manner as we have defined $\hat{\alpha}^t$.

Definition 3.5 (Correct α^μ) Let p be a selection predicate. A definition of α^μ is *correct* if the following properties hold: (i) $\alpha^\mu(p) \leftarrow p$; (ii) $\alpha^\mu(p) \leftarrow \bar{\alpha}^\mu(p)$; (iii) $\alpha^\mu(p) \leftarrow \bar{\alpha}^\mu(\alpha^\mu(p))$. □

Theorem 3.3 (Correctness & minimality of $\bar{\alpha}^\mu$) $\bar{\alpha}^\mu$ is a correct definition of α^μ. Furthermore, it is minimal: given any correct definition of α^μ, $\bar{\alpha}^\mu(p) \rightarrow \alpha^\mu(p)$ for any selection predicate p. □

Definition 3.6 ($\hat{\alpha}^\mu$) Let p be a selection predicate with negations pushed down to the level of atomic predicates. We define $\hat{\alpha}^\mu(p)$ inductively on the structure of p. If p is an atomic or negated atomic predicate, $\hat{\alpha}^\mu(p)$ is defined as a quantifier-free predicate logically equivalent to $\bar{\alpha}^\mu(p)$ (specified in Tables 3 and 4 of Appendix A). $\hat{\alpha}^\mu(p_1 \vee p_2)$ is defined as $\hat{\alpha}^\mu(p_1) \vee \hat{\alpha}^\mu(p_2)$. $\hat{\alpha}^\mu(p_1 \wedge p_2)$ is defined as $\hat{\alpha}^\mu(p_1) \wedge \hat{\alpha}^\mu(p_2)$. □

Theorem 3.4 (Correctness of $\hat{\alpha}^\mu$) $\hat{\alpha}^\mu$ is a correct definition of α^μ according to Def. 3.5. $\square$

For atomic predicates and their negations, $\hat{\alpha}^\mu$ is minimal. For composite predicates, minimality is preserved over disjunction, but not over conjunction. The reasoning is analogous to the reasoning for $\hat{\alpha}^t$, but with a small twist. Intuitively, if there exists an update μ_1 that makes p_1 true and there also exists an update μ_2 that makes p_2 true, we know that there exists an update that makes $p_1 \vee p_2$ true; either μ_1 or μ_2 will do. On the other hand, we cannot guarantee that there exists an update that makes $p_1 \wedge p_2$ true, because μ_1 may not make p_2 true and μ_2 may not make p_1 true.

Table 3 of Appendix A specifies $\hat{\alpha}^\mu$ for all atomic predicates and their negations, in the general case where T may be a set of time periods. Table 4 assumes that T contains only a single period. Because of the single-period constraint, the $\hat{\alpha}^\mu$ specifications in Table 4 are at least as selective as those in Table 3, if not more selective. The length of the update window $[\texttt{NOW} - l, \texttt{NOW}]$ also plays an important role. Many $\hat{\alpha}^\mu$ specifications contain references to l; a smaller l makes them more selective. In particular, $l = 0$ corresponds to the special case of nontemporal sources.

Example 3.3 To illustrate the importance of constraints, we explain $\hat{\alpha}^\mu(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ under different constraints. Note how $\hat{\alpha}^\mu$ becomes more and more selective as we add the single-period constraint and decrease the length of the update window.

Update window $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$*, no single-period constraint.* Note that $\texttt{NOW} - \texttt{3 days}$ is inside the update window. Thus, for any tuple r, we can always update the part of $r.\texttt{T}$ within the current update window to end at $t_{now} - \texttt{3 days}$. Therefore, $\hat{\alpha}^\mu(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ is true, as indicated by Row 22 of Table 3.

Update window $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$*, single-period constraint.* $\texttt{NOW} - \texttt{3 days}$ is still inside the update window, but we can no longer apply an arbitrary update since we need to make sure that T remains a single period afterwards. Specifically, if $r.\texttt{TE} < \texttt{NOW} - \texttt{8 days}$, we cannot modify $r.\texttt{T}$ because any update within $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$ would create another disjoint period in $r.\texttt{T}$. On the other hand, as long as $r.\texttt{TE} \geq \texttt{NOW} - \texttt{8 days}$, we can make $(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ true by updating the part of $r.\texttt{T}$ within the update window to be $[t_{now} - \texttt{7 days}, t_{now} - \texttt{3 days}]$. This update will not create more periods in $r.\texttt{T}$, because $[t_{now} - \texttt{7 days}, t_{now} - \texttt{3 days}]$ is connected to the part of $r.\texttt{T}$ before the update window, if any. Therefore, $\hat{\alpha}^\mu(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ is $(\texttt{TE} \geq \texttt{NOW} - \texttt{8 days})$, as indicated by Row 23 of Table 4.

Update window $[\texttt{NOW}, \texttt{NOW}]$*, single-period constraint.* Again, because of the single-period constraint, we can update a tuple r only if $r.\texttt{TE}$ lies within or next to the update window. Thus, all we can do is terminate $r.\texttt{T}$ right before the current update window, i.e., on $t_{now} - \texttt{1 day}$, which is still not enough to satisfy $(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$. Therefore, $\hat{\alpha}^\mu(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ remains $(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$, as indicated by Row 21 of Table 4. $\square$

3.3 Combining Time Advances And Data Updates: α

Finally, we tackle the problem of making a temporal selection view self-maintainable with respect to both time advances and data updates. For a view $\sigma_p(R)$, our goal is to derive $\alpha(p)$ by relaxing the original selection predicate p such that $\sigma_{\alpha(p)}(R)$ is a self-maintainable superview of $\sigma_p(R)$. Following the same path that we have taken when deriving α^t and α^μ, we first define $\bar{\alpha}$ as a lower bound for α.

Definition 3.7 **($\bar{\alpha}$)** Let t_{now} be the current time and p a selection predicate. Suppose that the update window is $[\texttt{NOW} - l, \texttt{NOW}]$. Define $\bar{\alpha}(p)$ as a selection predicate over a temporal tuple r:

$$\begin{aligned}(\bar{\alpha}(p))(r) \overset{\text{def}}{=} \; & \exists t_1 \geq t_{now} : \exists \mu_1 \text{ in } [t_1 - l, t_1] : \\ & \exists t_2 \geq t_1 : \exists \mu_2 \text{ in } [t_2 - l, t_2] : ... \\ & \exists t_n \geq t_{n-1} : \exists \mu_n \text{ in } [t_n - l, t_n] : \\ & \exists t_{n+1} \geq t_n : [p]_{t_{n+1}}([\mu_n]_{t_n}(...([\mu_2]_{t_2}([\mu_1]_{t_1}(r)))...))\end{aligned}$$

Here, $n \geq 0$, and each μ_i is an update of `T` that may possibly leave `T` unchanged but may not change `T` to $\varnothing$. □

The sequence of t_i's and μ_i's in the definition is interpreted as follows. First, time advances from t_{now} to t_1, and an update μ_1 is applied to r at t_1. Then, time advances from t_1 to t_2, and μ_2 is applied at t_2, etc. Finally, after μ_n has been applied at t_n, time advances to t_{n+1} and p becomes true. Intuitively, a tuple r satisfies $\bar{\alpha}(p)$ iff r will satisfy p after going through a finite sequence of interleaving time advances and data updates. The sequence is effectively empty when $n = 0$ and $t_1 = t_{now}$. Therefore, $\sigma_{\bar{\alpha}(p)}(R)$ contains the current contents of $\sigma_p(R)$ plus enough auxiliary information to maintain $\sigma_p(R)$, and nothing more. Using $\bar{\alpha}$, we now formally define what constitutes a correct α, and show that $\bar{\alpha}$ is a correct α as well as the minimal one.

Definition 3.8 **(Correct α)** Let p be a selection predicate. A definition of α is *correct* if the following properties hold: (i) $\alpha(p) \leftarrow p$; (ii) $\alpha(p) \leftarrow \bar{\alpha}(p)$; (iii) $\alpha(p) \leftarrow \bar{\alpha}(\alpha(p))$. □

Theorem 3.5 **(Correctness & minimality of $\bar{\alpha}$)** $\bar{\alpha}$ is a correct definition of α. Furthermore, it is minimal: given any correct definition of α, $\bar{\alpha}(p) \rightarrow \alpha(p)$ for any selection predicate p. □

Since Def. 3.7 uses an unbounded number of existential quantifiers, we need to find a computable alternative to $\bar{\alpha}$. We shall take the same approach as before, defining $\hat{\alpha}$ inductively starting from atomic predicates and their negations. However, even for a simple atomic or negated atomic predicate p, finding the quantifier-free predicate equivalent to $\bar{\alpha}(p)$ is a difficult and sometimes extremely tricky task. Fortunately, we can use the quantifier-free versions of $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$ derived in the previous subsections to compute the quantifier-free version of $\bar{\alpha}$, instead of deriving it directly from Def. 3.7. Now, the question becomes, how should we define $\bar{\alpha}$ in terms of $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$?

A first simple guess is $\bar{\alpha}(p) \leftrightarrow \bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$. Recall that $\sigma_{\bar{\alpha}^t(p)}(R)$ contains enough information to maintain $\sigma_p(R)$ with respect to time advances, and $\sigma_{\bar{\alpha}^\mu(p)}(R)$ contains enough information to maintain $\sigma_p(R)$ with respect to data updates. Therefore, $\sigma_{\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)}(R)$ contains enough information to maintain $\sigma_p(R)$ with respect to either time advances or data updates. This definition may seem reasonable, but it turns out to be incorrect, as illustrated by the following example.

Example 3.4 Let p be the predicate ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$). Suppose that the update window is $[\texttt{NOW}, \texttt{NOW}]$ and `T` may contain only a single period. This scenario was considered first in Example 2.1. As we have seen in Examples 3.2 and 3.3, respectively, $\bar{\alpha}^t(p)$ is ($\texttt{TE} \geq \texttt{NOW} - \texttt{3 days}) \wedge (\texttt{TE is not NOW}$), and $\bar{\alpha}^\mu(p)$ is ($\texttt{TE} = \texttt{NOW} - \texttt{3 days}$). Hence, $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$ is ($\texttt{TE} \geq \texttt{NOW} - \texttt{3 days}) \wedge (\texttt{TE is not NOW}$). However, $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$ may miss some tuples with the potential of satisfying p. Specifically, $r_5 = \langle \texttt{5}, [\texttt{09/25}, \texttt{NOW}] \rangle$ from Figure 1 does not satisfy $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$, but in Example 2.1 it was shown to be necessary for the maintenance of $\sigma_p(R)$.

The problem is that $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$ selects tuples that can satisfy p either through time advances or through data updates, but it fails to select tuples that can satisfy p through a sequence of interleaving time advances and data updates. For instance, as discussed in Example 2.1, r_5 satisfies p by going through a data update on `10/01` first, and then through an advance of time from `10/01` to `10/03`. Thus, r_5 is missed by $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$. Another way to see the problem is by observing that $\sigma_{\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)}(R)$ is not self-maintainable. In particular, the update on `10/01` will change r_5 to $r'_5 = \langle \mathtt{5}, [\mathtt{09/25}, \mathtt{09/30}] \rangle$, which satisfies $\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)$. Since $\sigma_{\bar{\alpha}^t(p) \vee \bar{\alpha}^\mu(p)}(R)$ does not contain r_5, it will be not be maintainable when this update occurs. □

The real source of the problem is that when we relax a selection predicate by $\bar{\alpha}^t$, we may introduce auxiliary data that is not maintainable with respect to data updates, and similarly, when we relax a selection predicate by $\bar{\alpha}^\mu$, we may introduce auxiliary data that is not maintainable with respect to time advances. This analysis suggests the following procedure for computing $\bar{\alpha}(p)$. Initially, we set the answer predicate p' to p. Of course, $\sigma_{p'}(R)$ may not be self-maintainable yet, so we relax p' using $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$, i.e., we replace p' with $\bar{\alpha}^\mu(\bar{\alpha}^t(p'))$. By relaxing p', we may have introduced more auxiliary data into $\sigma_{p'}(R)$ which needs to be maintained. Therefore, we must replace p' with $\bar{\alpha}^\mu(\bar{\alpha}^t(p'))$ again. This process is repeated until p' no longer changes, and at this point $\sigma_{p'}(R)$ is self-maintainable. Thm. 3.6 below indicates that $\bar{\alpha}(p)$ is indeed logically equivalent to p' as computed by the above procedure.

Theorem 3.6 (Fixed-point formulation of $\bar{\alpha}$) Let p be a selection predicate. $\bar{\alpha}(p) \leftrightarrow (\bar{\alpha}^\mu \circ \bar{\alpha}^t)^*(p)$, where $(\bar{\alpha}^\mu \circ \bar{\alpha}^t)^*(p)$ denotes the fixed point of applying $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$ to p. □

Thm. 3.6 provides us a way to compute $\bar{\alpha}(p)$, although a number of practical issues still remain. To evaluate $(\bar{\alpha}^\mu \circ \bar{\alpha}^t)^*(p)$, we must use $\hat{\alpha}^t$ and $\hat{\alpha}^\mu$, the computable alternatives to $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$ defined in Sections 3.1 and 3.2. As we have seen, $\hat{\alpha}^t$ and $\hat{\alpha}^\mu$ are not always equivalent to $\bar{\alpha}^t$ and $\bar{\alpha}^\mu$ for conjunctive predicates. The fixed-point iteration may introduce conjunctions into the relaxed predicate even if the original predicate is atomic. Thus, during the iteration, we may want to augment the relaxed predicate in order to improve the minimality of the result, using the technique introduced in Section 3.1.1. Moreover, testing the termination condition of the fixed-point iteration involves testing the logical equivalence of temporal predicates. Therefore, if we wish to automate the fixed-point computation, we will again need a theorem prover for our temporal language. Since we do not want to rely on the existence of such a theorem prover, we have computed $\bar{\alpha}$ manually for all atomic predicates and their negations in Tables 5 and 6 of Appendix A. Care has been taken to ensure that all results are indeed minimal.

Example 3.5 As an example of the fixed-point computation, we show how to use $\hat{\alpha}^t$ and $\hat{\alpha}^\mu$ to derive $\bar{\alpha}(p)$, where p is $(\mathtt{TE} = \mathtt{NOW} - \mathtt{3\ days})$. As in Example 3.4, we assume that the update window is $[\mathtt{NOW}, \mathtt{NOW}]$ and the single-period constraint holds. First, we apply $\hat{\alpha}^t$ to p. According to Row 16 of Table 2, $\hat{\alpha}^t(p)$ is:

$$(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days}) \wedge (\mathtt{TE\ is\ not\ NOW})$$

Next, we apply $\hat{\alpha}^\mu$ to $\hat{\alpha}^t(p)$. According to Rows 24 and 20 of Table 4, $\hat{\alpha}^\mu(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$ remains $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$, while $\hat{\alpha}^\mu(\mathtt{TE\ is\ not\ NOW})$ becomes $(\mathtt{TS} < \mathtt{NOW})$. Thus, $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$ is:

$$(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days}) \wedge (\mathtt{TS} < \mathtt{NOW})$$

Since this predicate is different from p, we must continue the fixed-point iteration. According to

Rows 18 and 10 of Table 2, $\hat{\alpha}^t(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$ remains $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$, while $\hat{\alpha}^\mu(\mathtt{TS} < \mathtt{NOW})$ becomes `true`. Thus, $\hat{\alpha}^t(\hat{\alpha}^\mu(\hat{\alpha}^t(p)))$ is:

$$(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$$

Subsequent applications of $\hat{\alpha}^\mu$ and $\hat{\alpha}^t$ all yield the same predicate. Therefore, $(\hat{\alpha}^\mu \circ \hat{\alpha}^t)^*(p)$ is $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$.

Notice that the steps in the fixed-point computation correspond to the sequence of interleaving time advances and data updates that may cause a tuple to satisfy the original predicate. For instance, suppose that the current time is `10/01` and there is a tuple r with $r.\mathtt{T} = [\mathtt{10/01}, \mathtt{NOW}]$. Currently, r satisfies $\hat{\alpha}^t(\hat{\alpha}^\mu(\hat{\alpha}^t(p)))$, but not $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$. The difference between $\hat{\alpha}^t(\hat{\alpha}^\mu(\hat{\alpha}^t(p)))$ and $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$ is an application of $\hat{\alpha}^t$, hinting that an advance of time will make r satisfy $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$. Indeed, when time advances to `10/02`, r satisfies $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$. However, r still does not satisfy $\hat{\alpha}^t(p)$. This time, $\hat{\alpha}^\mu(\hat{\alpha}^t(p))$ and $\hat{\alpha}^t(p)$ differ by an application of $\hat{\alpha}^\mu$, hinting that a data update is required. Indeed, this update subtracts $[\mathtt{10/02}, \mathtt{NOW}]$ from $r.\mathtt{T}$, changing it to $[\mathtt{10/01}, \mathtt{10/01}]$. Now, r satisfies $\hat{\alpha}^t(p)$. Finally, an advance of time from `10/02` to `10/04` causes r to satisfy p.

The above reasoning can be generalized to show that any tuple that currently satisfies $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$ will eventually satisfy $(\mathtt{TE} = \mathtt{NOW} - \mathtt{3\ days})$ after going through the sequence of time advances and data updates corresponding to the steps in the fixed-point computation. We have in effect a proof of the minimality of $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$; that is, $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$ is a quantifier-free predicate equivalent to $\bar{\alpha}(\mathtt{TE} = \mathtt{NOW} - \mathtt{3\ days})$. □

Table 5 in Appendix A lists the practical, quantifier-free version of $\bar{\alpha}$ for all atomic predicates and their negations, in the general case where `T` may be a set of time periods. Table 6 assumes that `T` contains only a single period. Effects of the single-period constraint and the length of the update window are clearly visible among the table entries. From these tables, we can proceed to define $\hat{\alpha}$. Like $\hat{\alpha}^t$ and $\hat{\alpha}^\mu$, $\hat{\alpha}$ is minimal for predicates with no conjunctions, but it does not preserve minimality over conjunction. If needed, we can still improve $\hat{\alpha}$ using the technique of augmenting predicates discussed in Section 3.1.1.

Definition 3.9 ($\hat{\alpha}$) Let p be a selection predicate with negations pushed down to the level of atomic predicates. We define $\hat{\alpha}(p)$ inductively on the structure of p. If p is an atomic or negated atomic predicate, $\hat{\alpha}(p)$ is defined as a quantifier-free predicate logically equivalent to $\bar{\alpha}^t(p)$ (specified in Tables 5 and 6 of Appendix A). $\hat{\alpha}(p_1 \vee p_2)$ is defined as $\hat{\alpha}(p_1) \vee \hat{\alpha}(p_2)$. $\hat{\alpha}(p_1 \wedge p_2)$ is defined as $\hat{\alpha}(p_1) \wedge \hat{\alpha}(p_2)$. □

Theorem 3.7 (Correctness of $\hat{\alpha}$) $\hat{\alpha}$ is a correct definition of α according to Def. 3.8. □

Example 3.6 To conclude this subsection, we derive superviews for all example views in Section 2.3. For Examples 2.1–2.4, we also verify that the superviews indeed contain all and only those tuples of Figures 1 and 2 required for self-maintenance as identified by our analysis in Section 2.3.

Example 2.1. With the update window $[\mathtt{NOW}, \mathtt{NOW}]$ and the single-period constraint, $\hat{\alpha}(\mathtt{TE} = \mathtt{NOW} - \mathtt{3\ days})$ is $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days})$ according to Row 21 of Table 6. Hence, the superview of V is $\sigma_{\mathtt{TE} \geq \mathtt{NOW} - \mathtt{3\ days}}(R)$, which contains r_3, r_4, and r_5.

Example 2.2. With the update window $[\mathtt{NOW} - \mathtt{7\ days}, \mathtt{NOW}]$ and the single-period constraint, $\hat{\alpha}(\mathtt{TE} = \mathtt{NOW} - \mathtt{3\ days})$ is $(\mathtt{TE} \geq \mathtt{NOW} - \mathtt{8\ days})$ according to Row 22 of Table 6. Hence, the superview of V is $\sigma_{\mathtt{TE} \geq \mathtt{NOW} - \mathtt{8\ days}}(R)$, which contains r_2, r_3, r_4, and r_5.

Example 2.3. With the update window $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$ and without the single-period constraint, $\hat{\alpha}(\texttt{TE} = \texttt{NOW} - \texttt{3 days})$ is $\texttt{true}$ according to Row 21 of Table 5. Hence, the superview of V is R.

Example 2.4. With the update window $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$ and without the single-period constraint, $\hat{\alpha}(\texttt{Dept} = \texttt{sales})$ remains $(\texttt{Dept} = \texttt{sales})$ according to Row 1 of Table 5, and $\hat{\alpha}(\texttt{length(T)} < \texttt{5 days})$ is $(\texttt{length(T} - [\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]) < \texttt{5 days})$ according to Row 28 of Table 5. Therefore, $\hat{\alpha}(\texttt{Dept} = \texttt{sales} \wedge \texttt{length(T)} < \texttt{5 days})$ is: $(\texttt{Dept} = \texttt{sales}) \wedge (\texttt{length(T} - [\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]) < \texttt{5 days})$. The superview of U is $\sigma_{\texttt{Dept=sales} \,\wedge\, \texttt{length(T}-[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]) < \texttt{5 days}}(S)$, which contains s_2 and s_3.

Example 2.5. With the update window $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$ and the single-period constraint, $\hat{\alpha}(\texttt{TE} = t)$ is exactly q according to Row 16 of Table 6. Hence, the superview of W is $\sigma_q(R)$. $\square$

3.4 Maintaining Selection Views

In the previous subsections we have described how to derive a self-maintainable superview for a temporal selection view. In this subsection we present an algorithm to maintain the original view and the superview incrementally. First, we introduce a semijoin operator $\ltimes$ over explicit attributes defined as: $R \ltimes S \stackrel{\text{def}}{=} \{r \mid r \in R \wedge (\exists s \in S : \forall A \in A_R \cap A_S : r.A = s.A)\}$. Like the usual relational semijoin, $R \ltimes S$ returns all R tuples that agree with some S tuple on all explicit attributes common to R and S. We define the antisemijoin operator $\overline{\ltimes}$ as: $R \,\overline{\ltimes}\, S \stackrel{\text{def}}{=} R - (R \ltimes S)$.

Algorithm 3.1, the incremental maintenance algorithm for selection views, is shown in Figure 3. There are two main cases in Algorithm 3.1: change propagation for data updates and view refresh for time advances. For change propagation, we distinguish two types of source changes and assume that they are reported to the warehouse in separate batches: $\triangle R$ contains new insertions to the temporal base relation R; $\triangledown\!\!\!\!\triangledown R$ and $\blacktriangle R$ together describe updates to existing tuples in R and deletions from R. It is possible to lift the assumption that sources can report the two types of changes separately, but doing so complicates the superview derivation and maintenance algorithms, especially when $\texttt{T}$ can be a set of time periods.

Each case of Algorithm 3.1 has two sets of equations arranged in columns. The left column shows how to maintain the superview incrementally; the right column shows how to maintain the original view incrementally. Let us focus on the left column first. The intuition is as follows.

In Case 1.1, given new insertions to the temporal base relation, we select all new tuples that satisfy the selection predicate of the superview and insert them into the superview.

In Case 1.2, given updates to the temporal base relation, we first identify all tuples in the superview that are affected by the updates, and store their original values in $\triangledown' V^\alpha$. We ignore updates to tuples not currently in the superview since by definition those tuples cannot possibly contribute to the superview. Next, we apply the updates to tuples in $\triangledown' V^\alpha$, and store the updated values in $\triangle' V^\alpha$. After the updates, some tuples may no longer satisfy the selection predicate of the superview; we delete them from V^α using $\triangledown V^\alpha$. Other tuples still satisfy the predicate after the updates; we update them in V^α using $\triangledown\!\!\!\!\triangledown V^\alpha$ and $\blacktriangle V^\alpha$. The second term of the union in $\triangledown\!\!\!\!\triangledown V^\alpha$ handles deletions from the temporal base relation.

In Case 2, given a request to refresh the view with respect to time, we re-evaluate the selection predicate of the superview and delete those tuples that no longer satisfy the predicate because of the advance of time. Again, we do not have to worry about tuples not currently in the superview

Algorithm 3.1 The original view is $V \stackrel{\text{def}}{=} \sigma_p(R)$ and the update window is w. The superview is $V^\alpha \stackrel{\text{def}}{=} \sigma_{\hat{\alpha}(p)}(R)$.

- Case 1: Change propagation.
  - Case 1.1: Warehouse receives $\triangle R$ in w.

$$
\begin{array}{rcl|rcl}
\triangle V^\alpha & = & \sigma_{\hat{\alpha}(p)}(\triangle R) & \triangle V & = & \sigma_p(\triangle R) \\
V^\alpha & \leftarrow & V^\alpha \cup \triangle V^\alpha & V & \leftarrow & V \cup \triangle V
\end{array}
$$

  - Case 1.2: Warehouse receives $\triangledown\!\!\!\!\!\mu R$ and $\triangle\!\!\!\!\!\mu R$ in w.

$$
\begin{array}{rcl|rcl}
\triangledown' V^\alpha & = & V^\alpha \ltimes (\triangledown\!\!\!\!\!\mu R \cup \triangle\!\!\!\!\!\mu R) & \triangledown' V & = & V \ltimes (\triangledown\!\!\!\!\!\mu R \cup \triangle\!\!\!\!\!\mu R) \\
\triangle' V^\alpha & = & \triangledown' V^\alpha - \triangledown\!\!\!\!\!\mu R \cup \triangle\!\!\!\!\!\mu R & \triangle' V & = & \triangledown' V - \triangledown\!\!\!\!\!\mu R \cup \triangle\!\!\!\!\!\mu R \\
\triangledown V^\alpha & = & \triangledown' V^\alpha \ltimes \sigma_{\neg\hat{\alpha}(p)}(\triangle' V^\alpha) & \triangledown V & = & \triangledown' V \ltimes \sigma_{\neg p}(\triangle' V) \\
 & & & \triangle V & = & \sigma_p(\triangle' V^\alpha) \overline{\ltimes} V \\
\triangledown\!\!\!\!\!\mu V^\alpha & = & \triangledown\!\!\!\!\!\mu R \ltimes \sigma_{\hat{\alpha}(p)}(\triangle' V^\alpha) & \triangledown\!\!\!\!\!\mu V & = & \triangledown\!\!\!\!\!\mu R \ltimes \sigma_p(\triangle' V) \\
 & \cup & \triangledown\!\!\!\!\!\mu R \overline{\ltimes} \triangle' V^\alpha & & \cup & \triangledown\!\!\!\!\!\mu R \overline{\ltimes} \triangle' V \\
\triangle\!\!\!\!\!\mu V^\alpha & = & \triangle\!\!\!\!\!\mu R \ltimes \sigma_{\hat{\alpha}(p)}(\triangle' V^\alpha) & \triangle\!\!\!\!\!\mu V & = & \triangle\!\!\!\!\!\mu R \ltimes \sigma_p(\triangle' V) \\
V^\alpha & \leftarrow & V^\alpha - \triangledown V^\alpha - \triangledown\!\!\!\!\!\mu V^\alpha \cup \triangle\!\!\!\!\!\mu V^\alpha & V & \leftarrow & V - \triangledown V \cup \triangle V - \triangledown\!\!\!\!\!\mu V \cup \triangle\!\!\!\!\!\mu V
\end{array}
$$

- Case 2: View refresh.

$$
\begin{array}{rcl|rcl}
\triangledown V^\alpha & = & \sigma_{\neg\hat{\alpha}(p)}(V^\alpha) & \triangledown V & = & \sigma_{\neg p}(V) \\
 & & & \triangle V & = & \sigma_p(V^\alpha) \overline{\ltimes} V \\
V^\alpha & \leftarrow & V^\alpha - \triangledown V^\alpha & V & \leftarrow & V - \triangledown V \cup \triangle V
\end{array}
$$

Figure 3: Incremental maintenance algorithm for selection views.

since they cannot possibly contribute to the superview.

Note that Algorithm 3.1 accesses only contents of the source changes and the superview itself in order to maintain the superview. This observation confirms the fact that the superview is self-maintainable.

For completeness, we also provide the maintenance equations for the original view in the right column of Algorithm 3.1. Usually there is no need to compute these equations because we can simply store the superview instead of the original view at the warehouse. However, these equations are useful if we decide to materialize the original view in addition to the superview for performance. Moreover, they may be helpful for the maintenance of other warehouse views defined over the original view.

The intuition for maintaining the original view is analogous to the intuition for maintaining the superview. The only notable difference is that Cases 1.2 and 2 generate $\triangle V$, but not $\triangle V^\alpha$. In Case 1.2, $\triangle V$ contains the tuples in V^α that satisfy p after the data updates. In Case 2, $\triangle V$ contains the tuples in V^α that satisfy p after the time advance. $\triangle V^\alpha$ is not generated because V^α by definition already contains all tuples that could potentially satisfy $\hat{\alpha}(p)$ after any data updates and/or time advances.

It is helpful to determine the properties of the incremental changes generated by Algorithm 3.1 since we may need to propagate these changes further to other views defined over V. $\triangledown\!\!\!\!\!\mu V$ and

---

Algorithm 3.2 The original view is $V \stackrel{\text{def}}{=} \sigma_p(R)$, where p has no references to `T` or `NOW`. The update window is w. The superview is identical to V.

- Case 1: Change propagation.
  - Case 1.1: Warehouse receives $\triangle R$ in w.

$$
\begin{aligned}
\triangle V &= \sigma_p(\triangle R) \\
V &\leftarrow V \cup \triangle V
\end{aligned}
$$

  - Case 1.2: Warehouse receives $\bar{\triangledown} R$ and $\bar{\triangle} R$ in w.

$$
\begin{aligned}
\bar{\triangledown} V &= \sigma_p(\bar{\triangledown} R) \\
\bar{\triangle} V &= \sigma_p(\bar{\triangle} R) \\
V &\leftarrow V - \bar{\triangledown} V \cup \bar{\triangle} V
\end{aligned}
$$

- Case 2: View refresh. Do nothing.

Figure 4: Improved maintenance algorithm for selection views with nontemporal predicates.

$\bar{\triangle} V$ are updates to existing tuples in V. $\triangledown V$ contains "pure" deletions from V, i.e., $\triangledown V$ always deletes tuples from V completely, instead of only shrinking their time attributes. $\triangle V$ contains "pure" insertions into V, i.e., $\triangle V$ never expands the time attribute of any existing tuple in V. Furthermore, $\bar{\triangledown} V$ and $\bar{\triangle} V$ are always inside the update window w; $\triangledown V$ may be outside w; $\triangle V$ generated by Case 1.1 is always inside w, but $\triangle V$ generated by Cases 1.2 and 1.3 may be outside w.

3.4.1 Maintaining Selection Views With Nontemporal Predicates

Consider the special case where the selection predicate p does not reference `T` or `NOW`. Note that although the predicate is not temporal, the view still is. According to Row 1 of Tables 5 and 6, $\hat{\alpha}(p)$ is the same as p, so the superview is the same as the original view. Algorithm 3.1 still works, but Cases 1.2 and 2 are inefficient. Therefore, we have devised Algorithm 3.2 shown in Figure 4, which is optimized for selection views with nontemporal predicates.

Several differences between Algorithms 3.2 and 3.1 are worth noting. To compute the incremental changes to the view, Algorithm 3.2 uses only the contents of the source changes, while Algorithm 3.1 also uses the contents of the superview. Algorithm 3.2 does not generate $\triangledown V$ or $\triangle V$ in Cases 1.2 and 2, while Algorithm 3.1 may. The reason for these differences is that neither data updates nor time advances can affect the truth value of a nontemporal predicate. $\triangle V$, $\bar{\triangledown} V$, and $\bar{\triangle} V$ generated by Algorithm 3.2 are all inside the update window w.

4 Temporal Join Views

In this section we study the problem of making a temporal join view self-maintainable. Given a temporal join view $V \stackrel{\text{def}}{=} \bowtie_p(R_1, ..., R_i, ..., R_n)$, we first push local selection conditions down, resulting in the form $\bowtie_{jp}(V_1, ..., V_i, ..., V_n)$, where $V_i \stackrel{\text{def}}{=} \sigma_{sp_i}(R_i)$. For each local selection view V_i,

Algorithm 4.1 Given a temporal join view $V \stackrel{\text{def}}{=} \bowtie_p(R_1, ..., R_i, ..., R_n)$, derive a set of auxiliary views $\mathcal{X}$ such that $\mathcal{X}$ and V together are self-maintainable. The update window of R_i is w_i, $1 \leq i \leq n$.

METHOD: First, push down all local selection conditions to the base relations in V, resulting in the form $\bowtie_{jp}(V_1, ..., V_i, ..., V_n)$, where $V_i \stackrel{\text{def}}{=} \sigma_{sp_i}(R_i)$. Let $w = \bigcup_{1 \leq i \leq n} w_i$ and let $\mathcal{X} = \varnothing$.

- If p has no references to `T` or `NOW`, then for each i, add $V_i^\alpha \stackrel{\text{def}}{=} \mathsf{FR}_w(\sigma_{sp_i}(R_i))$ to $\mathcal{X}$.
- If p references `T` and/or `NOW`, then for each i, add $V_i^\alpha \stackrel{\text{def}}{=} \sigma_{\hat{\alpha}(sp_i)}(R_i)$ to $\mathcal{X}$.

Figure 5: Deriving auxiliary views for join views.

we create an auxiliary view V_i^α. This overall strategy is similar to the one used to make nontemporal join views self-maintainable in [HZ96]. However, there are two important differences in the choice of the auxiliary views. First, in the nontemporal case of [HZ96], the auxiliary views are defined simply as the local selection views, and these views are always self-maintainable. In our case, we cannot always define V_i^α as V_i, because a temporal selection view may not be self-maintainable (Section 3). Second, in the temporal case, the update-window constraint provides a unique opportunity for optimizing auxiliary views. Sometimes it is possible to discard history outside the update window, thereby saving a substantial amount of storage. To formalize the operation of discarding history, we introduce a *fragment operator* FR defined as: $\mathsf{FR}_w(R) \stackrel{\text{def}}{=} \{\langle r.A, r.\mathtt{T} \cap w\rangle \mid r \in R \wedge r.\mathtt{T} \cap w \neq \varnothing\}$, where w is a time period. In other words, $\mathsf{FR}_w(R)$ returns the fragment of R inside w and ignores the history outside w.

Algorithm 4.1, shown in Figure 5, specifies how to derive auxiliary views in order to make a temporal join view self-maintainable. If jp and all of the sp_i's are nontemporal (the first bullet in Figure 5), then we only need the fragment of each V_i inside w, the largest of the update windows. The reason is three-fold. First, to be able to maintain V, each V_i^α must contain enough information so we can join with updates to the other V_j's. Since all sp_j's are nontemporal, all updates to V_j's are guaranteed to be inside w (Section 3.4.1). Since jp is nontemporal, we do not need the time attribute to evaluate jp; only the part of the time attribute inside w is needed to compute the time attribute of the join result. Second, the V_i^α's must contain enough information so we can refresh the join view as time advances. This requirement is satisfied trivially because we do not need to worry about time advances when jp is nontemporal. Third, we must ensure that the V_i^α's themselves are self-maintainable. They indeed are, as each V_i^α can be maintained by Algorithm 4.3 using only the data in V_i^α itself.

On the other hand, if jp or any sp_j is temporal (the second bullet in Figure 5), we need at least each V_i in its entirety. In the case where some sp_j is temporal, the entire V_i is required to join with updates to V_j, which may be outside the update window because of the temporal selection predicate (Section 3.4). In the case where jp is temporal, the entire V_i may be required to refresh the join as time advances. Furthermore, when an sp_i is temporal, we need the entire superview of V_i in order to ensure self-maintainability. This intuition is properly encoded by Algorithm 4.1 because $\hat{\alpha}(sp_i) \leftrightarrow sp_i$ when sp_i is nontemporal.

Lastly, note that the auxiliary views specified by Algorithm 4.1 are not guaranteed to be minimal, as our next example illustrates. We plan to improve Algorithm 4.1 as future work.

Example 4.1 Recall from Example 2.4 the temporal base relation $S(\texttt{Emp}, \texttt{Dept}, \texttt{T})$, which records the set of days worked by a given employee in a given department. Suppose another temporal base relation $Q(\texttt{Emp}, \texttt{Supr}, \texttt{T})$ records the history of all employee-supervisor relationships. The update window for both S and Q is $[\texttt{NOW} - \texttt{7 days}, \texttt{NOW}]$.

Nontemporal predicate; minimal auxiliary views. Consider a temporal join view with a nontemporal predicate: $J_1 \stackrel{\text{def}}{=} \bowtie_{S.\texttt{Emp}=Q.\texttt{Emp} \wedge Q.\texttt{Supr}=\texttt{Mary}}(S, Q)$, which keeps track of the work history of all employees during the time that they are supervised by Mary. Let us apply Algorithm 4.1 to make J_1 self-maintainable. After all local selection conditions have been pushed down, J_1 becomes $\bowtie_{S.\texttt{Emp}=Q.\texttt{Emp}}(V_{S1}, V_{Q1})$, where:

$$V_{S1} \stackrel{\text{def}}{=} S \qquad V_{Q1} \stackrel{\text{def}}{=} \sigma_{\texttt{Supr}=\texttt{Mary}}(Q)$$

Since all predicates are nontemporal, the auxiliary views are:

$$V_{S1}^{\alpha} \stackrel{\text{def}}{=} \mathsf{FR}_{[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]}(S) \qquad V_{Q1}^{\alpha} \stackrel{\text{def}}{=} \mathsf{FR}_{[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]}(\sigma_{\texttt{Supr}=\texttt{Mary}}(Q))$$

Temporal predicate; obviously nonminimal auxiliary views. We can modify J_1 to show that Algorithm 4.1 does not always produce minimal auxiliary views. Suppose that we add a conjunct $(S.\texttt{T}\ \texttt{overlaps}\ Q.\texttt{T})$ to the join predicate. This additional conjunct does not change the meaning of J_1, because time attributes of joining tuples must overlap by the definition of temporal join (Section 2.1). However, seeing that the new join predicate references T, Algorithm 4.1 would produce V_Q^{α} as $\sigma_{\texttt{Supr}=\texttt{Mary}}(Q)$ instead of $\mathsf{FR}_{[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]}(\sigma_{\texttt{Supr}=\texttt{Mary}}(Q))$.

Temporal predicate; subtly nonminimal auxiliary views. Suppose now we are only interested in the inexperienced employees supervised by Mary. For this purpose, we define:

$$J_2 \stackrel{\text{def}}{=} \bowtie_{\texttt{length}(S.\texttt{T}) < \texttt{5 days} \wedge S.\texttt{Emp}=Q.\texttt{Emp} \wedge Q.\texttt{Supr}=\texttt{Mary}}(S, Q)$$

With local selections pushed down, J_2 becomes $\bowtie_{S.\texttt{Emp}=Q.\texttt{Emp}}(V_{S2}, V_{Q2})$, where:

$$V_{S2} \stackrel{\text{def}}{=} \sigma_{\texttt{length(T)} < \texttt{5 days}}(S) \qquad V_{Q2} \stackrel{\text{def}}{=} \sigma_{\texttt{Supr}=\texttt{Mary}}(Q)$$

Since the selection predicate of V_{S2} references T, the auxiliary views are the superviews of V_{S2} and V_{Q2}, according to Algorithm 4.1:

$$V_{S2}^{\alpha} \stackrel{\text{def}}{=} \sigma_{\texttt{length(T}-[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]) < \texttt{5 days}}(S) \qquad V_{Q2}^{\alpha} \stackrel{\text{def}}{=} \sigma_{\texttt{Supr}=\texttt{Mary}}(Q)$$

Here, we may need the fragment of V_{Q2} outside the update window because the updates to V_{S2} may be outside the update window. However, it turns out that we still can do better than storing V_{Q2} in its entirety. Notice that if the updates to V_{S2} are outside the update window, then they must come from tuples in V_{S2}^{α}. Therefore, in addition to the fragment of V_{Q2} inside the update window, we only need the fragments of V_{Q2} that join with V_{S2}^{α}:

$$V_{Q2}^{\alpha'} \stackrel{\text{def}}{=} \mathsf{FR}_{[\texttt{NOW}-\texttt{7 days},\texttt{NOW}]}(\sigma_{\texttt{Supr}=\texttt{Mary}}(Q)) \cup \pi_{\{\texttt{Emp},\texttt{Supr}\}}(\bowtie_{S.\texttt{Emp}=Q.\texttt{Emp}}(V_{S2}^{\alpha}, \sigma_{\texttt{Supr}=\texttt{Mary}}(Q)))$$

$V_{Q2}^{\alpha'}$ is a smaller auxiliary view than V_{Q2}^{α} derived by Algorithm 4.1. Furthermore, $V_{Q2}^{\alpha'}$ is self-maintainable despite its complex definition. □

Next we consider how to incrementally maintain a temporal join view together with the auxiliary views derived by Algorithm 4.1. See Algorithm 4.2 in Figure 6. In the case where p is temporal, we can redefine the original view as a join of the auxiliary views. Because all auxiliary views are materialized, we can then maintain the original view using the standard maintenance expressions for temporal join from [YW98a], which may access the entire contents of the join operands. Alternatively, we may choose not to maintain the original view at all, since it can be recomputed from the auxiliary views as needed. As for the auxiliary views themselves, they can be maintained using Algorithm 3.1.

In the case where p is nontemporal, the original view cannot be redefined in terms of the

Algorithm 4.2 The original join view is $V \stackrel{\text{def}}{=} \bowtie_p(R_1, ..., R_i, ..., R_n)$. With local selection conditions pushed down, V has the form $\bowtie_{jp}(V_1, ..., V_i, ..., V_n)$, where $V_i \stackrel{\text{def}}{=} \sigma_{sp_i}(R_i)$. The set of auxiliary views produced for V by Algorithm 4.1 is $\mathcal{X} = \{V_1^\alpha, ..., V_i^\alpha, ..., V_n^\alpha\}$. The update window of R_i is w_i, $1 \le i \le n$, and $w = \bigcup_{1 \le i \le n} w_i$.
METHOD: If p references `T` and/or `NOW`, V can be redefined in terms of the auxiliary views as: $\bowtie_{jp \wedge sp_1 \wedge ... \wedge sp_i \wedge ... \wedge sp_n}(V_1^\alpha, ..., V_i^\alpha, ..., V_n^\alpha)$. Each V_i^α is the superview of V_i and can be maintained using Algorithm 3.1. Since all V_i^α's are materialized, V can be maintained using the maintenance expressions from [YW98a].
If p has no references to `T` or `NOW`:

- Case 1: Change propagation. Warehouse receives either $\triangle R_i$, or $\triangledown R_i$ and $\blacktriangle R_i$, in w_i. Use Algorithm 4.3 to maintain V_i^α, and use Algorithm 3.2 to compute $\triangle V_i$, $\triangledown V_i$, and $\blacktriangle V_i$ as incremental changes to V_i. Then:

$$
\begin{array}{rcl}
\triangle V & = & \bowtie_{jp}(V_1^\alpha, ..., \triangle V_i, ..., V_n^\alpha) \\
\triangledown V & = & \bowtie_{jp}(V_1^\alpha, ..., \triangledown V_i, ..., V_n^\alpha) \\
\blacktriangle V & = & \bowtie_{jp}(V_1^\alpha, ..., \blacktriangle V_i, ..., V_n^\alpha) \\
V & \leftarrow & V \cup \triangle V - \triangledown V \cup \blacktriangle V
\end{array}
$$

- Case 2: View refresh. Use Algorithm 4.3 to maintain each V_i^α. Do nothing for V.

Figure 6: Incremental maintenance algorithm for join views and their auxiliary views.

auxiliary views, because each auxiliary view contains only a fragment of the selection view. In this case, we must ensure that the maintenance expressions only access the available fragments within the auxiliary views. The auxiliary views are maintained using Algorithm 4.3 in Figure 7. (Algorithm 3.1 is not applicable here because the auxiliary views contain the `FR` operator.) Once we have computed any change to an auxiliary view using Algorithm 4.3, we propagate the change to the original view as specified in the first bullet of Algorithm 4.2. Note that in the second bullet, view refresh is not needed for the original view since p is nontemporal and its truth value cannot change over time. Nevertheless, view refresh is still necessary for the auxiliary views since their fragment windows move as time advances.

5 Snapshot-Reducible Temporal Views

Temporal operators that contain no explicit references to time are *snapshot-reducible* [Sno87], which include σ_p, $\bowtie_p$, π, $-$, and $\cup$, where p contains no reference to `T` or `NOW`. A temporal view is snapshot-reducible if it is defined using only snapshot-reducible operators. Each snapshot-reducible operator corresponds closely to its counterpart in nontemporal relational algebra. Suppose that a temporal relation R represents the history of a nontemporal relation R^{nt}. Let $\tau_t(R)$ denote the state of R^{nt} at time instant t. Then, for a snapshot-reducible operator op and its nontemporal counterpart op^{nt}, $\tau_t(op(R_1, ..., R_n)) = op^{nt}(\tau_t(R_1), ..., \tau_t(R_n))$. It follows that a snapshot-reducible temporal view over temporal relations can be considered as the history of a nontemporal view over nontemporal relations. Formally, given a snapshot-reducible temporal view V defined over temporal relations $R_1, ..., R_n$, we can define a nontemporal view V^{nt} by replacing, in the definition

Algorithm 4.3 The auxiliary view is $V^\alpha \stackrel{\text{def}}{=} \mathsf{FR}_w(\sigma_p(R))$, where p has no references to `T` or `NOW`. The update window of R is w', not necessarily the same as w.

- Case 1: Change propagation.
  - Case 1.1: Warehouse receives $\triangle R$ in w'.

$$\begin{array}{rcl} \triangle V^\alpha & = & \mathsf{FR}_w(\sigma_p(\triangle R)) \\ V^\alpha & \leftarrow & V^\alpha \cup \triangle V^\alpha \end{array}$$

  - Case 1.2: Warehouse receives $\triangledown R$ and $\blacktriangle R$ in w'.

$$\begin{array}{rcl} \triangledown V^\alpha & = & \mathsf{FR}_w(\sigma_p(\triangledown R)) \\ \blacktriangle V^\alpha & = & \mathsf{FR}_w(\sigma_p(\blacktriangle R)) \\ V^\alpha & \leftarrow & V^\alpha - \triangledown V^\alpha \cup \blacktriangle V^\alpha \end{array}$$

- Case 2: View refresh.

$$\begin{array}{rcl} \triangledown V^\alpha & = & V^\alpha - \mathsf{FR}_w(V^\alpha) \\ V^\alpha & \leftarrow & V^\alpha - \triangledown V^\alpha \end{array}$$

Figure 7: Incremental maintenance algorithm for auxiliary views with the fragment operator.

of V, each snapshot-reducible operator *op* by its nontemporal counterpart op^{nt}, and each temporal relation R_i by a nontemporal relation R_i^{nt}. Suppose that each R_i records the history of R_i^{nt}. Then, V records the history of V^{nt}.

Snapshot-reducibility has several important implications. First, the contents of a snapshot-reducible view are affected only by data updates, not by time advances [YW98a]. Therefore, the maintenance algorithm for snapshot-reducible views need not perform view refresh. Second, snapshot-reducible operators preserve equalities in nontemporal relational algebra [Sno87]. In particular, we have shown in [YW98a] how to derive change propagation equations for snapshot-reducible views from the corresponding equations for nontemporal view maintenance, e.g., [QW91]. Third, snapshot-reducible operators preserve update-window constraints. That is, given a snapshot-reducible view V defined over $R_1, ..., R_n$, if R_i is updated inside window w_i, then the incremental changes to V induced by this update will also be in w_i, for the following reason. The update of R_i inside w_i does not alter the history of R_i^{nt} outside w_i. Hence, the states of V^{nt} outside w_i remain unchanged. Since V records the history of V^{nt}, the fragment of V outside w_i must remain unchanged as well; any incremental changes to V must be inside w_i.

The close relationship between V and V^{nt} in the snapshot-reducible case leads to the following algorithm for making V self-maintainable. The main idea is to reduce the self-maintenance problem to its nontemporal version. First, we make V^{nt} self-maintainable using known algorithms for nontemporal views, e.g., [HZ96, QGMW96]. Let $\mathcal{X}^{nt}$ denote the set of nontemporal auxiliary views derived by the nontemporal algorithm. For each X^{nt} in $\mathcal{X}^{nt}$, we define a temporal view X by replacing, in X^{nt}, each nontemporal operator op^{nt} by its snapshot-reducible temporal counterpart *op*, and each nontemporal relation R_i^{nt} by the temporal relation R_i. Suppose that the largest source update window is w. Then, the set of temporal auxiliary views for making V self-maintainable is $\mathcal{X} \stackrel{\text{def}}{=} \{\mathsf{FR}_w(X) \mid X^{nt} \in \mathcal{X}^{nt}\}$. We store only the fragment of X within w because snapshot-reducible operators preserve update-window constraints. To maintain V and $\mathcal{X}$ incrementally, we can reuse the change propagation equations for V^{nt} and $\mathcal{X}^{nt}$ produced by the nontemporal algorithm, with all operators, relations, and views replaced by their temporal counterparts. View refresh is not necessary for snapshot-reducible views, as discussed above. View refresh for auxiliary views with the fragment operator is handled by Case 2 of Algorithm 4.3. Notice that Algorithms 3.1, 4.1, and 4.2, when applied to snapshot-reducible views, reduce to exactly the algorithm given above (based on the nontemporal algorithm from [HZ96]); Algorithm 3.2 is a special case of the above algorithm for snapshot-reducible selection views.

6 Temporal Aggregate Views

We now consider the self-maintenance problem for a class of commonly used temporal aggregates called *moving-window aggregates* [NA89]. We use $\Pi_{A,l'}$ to denote the temporal aggregate operator, where A is the set of group-by and aggregated attributes, and $l' \geq 0$ specifies the length of the *aggregate window*. To formally define $\Pi_{A,l'}$, we extend the snapshot operator τ_t introduced in Section 5 with an additional length parameter, so that $\tau_{t,l'}(R)$ returns a nontemporal relation containing all tuples that were valid in R^{nt} at some point within $[t-l', t]$. Then, we define $\Pi_{A,l'}(R)$ to be a temporal relation with the property that $\tau_t(\Pi_{A,l'}(R)) = \Pi_A^{nt}(\tau_{t,l'}(R))$ for any time instant t, where Π_A^{nt} is the standard nontemporal grouping and aggregate operator. In other words, the value of the aggregate at time t is computed over all tuples valid at any point inside the aggregate window $[t-l', t]$. Hence, a temporal aggregate view can be seen as a history of aggregate values computed by moving the aggregate window along the time line.

The self-maintenance algorithm for temporal aggregates breaks down into three cases:

1. *Instantaneous aggregates.* A temporal aggregate with $l' = 0$ is termed *instantaneous* because the value of $\Pi_{A,0}(R)$ at a particular time instant depends only on the state of R^{nt} at that instant. $\Pi_{A,0}$ is snapshot-reducible. Thus, using the algorithm of Section 5, we can reduce the self-maintenance problem for instantaneous temporal aggregates to the self-maintenance problem for nontemporal aggregates considered in [AJB98]. The details are as follows. Suppose the update window of R is w. If all aggregate functions in A are incrementally maintainable, such as SQL `SUM`, `COUNT`, and `AVG` (which can be maintained as `SUM/COUNT`), then $\Pi_{A,0}(R)$ is self-maintainable and no auxiliary views are necessary. If some aggregate function in A is not always incrementally maintainable, such as `MIN` or `MAX`, then $\mathsf{FR}_w(R)$ is needed as an auxiliary view.

2. *Cumulative aggregates.* A temporal aggregate with $l' > 0$ is termed *cumulative* because the value of $\Pi_{A,l'}(R)$ at time t depends on the fragment of R within the aggregate window $[t-l', t]$. In this case, $\Pi_{A,l'}$ is not snapshot-reducible, so reduction to the nontemporal self-maintenance problem does not work. Suppose the update window of R is $w = [\mathtt{NOW} - l, \mathtt{NOW}]$. To make $\Pi_{A,l'}(R)$ self-maintainable, we need $\mathsf{FR}_{[\mathtt{NOW}-l-l',\mathtt{NOW}]}(R)$ as an auxiliary view. The intuition is that any aggregate value within the update window may be affected by a source update, and in general, updating an aggregate value at t requires $\mathsf{FR}_{[t-l',t]}(R)$ whether or not the aggregate function is incrementally maintainable. For example, recall Figure 2. Say that we need to update a temporal aggregate view $\Pi_{\{\mathtt{Dept},\mathtt{COUNT(*)}\},\mathtt{30\,days}}(S)$ given a source update $\triangledown S = \{\langle \mathtt{D}, \mathtt{sales}, [\mathtt{09/28}, \mathtt{NOW}]\rangle\}$. The count for `Dept` = `sales` at `NOW` should be decremented only if $\triangledown S$ completely removes the part of $s_5.\mathtt{T}$ within the aggregate window $[\mathtt{NOW} - \mathtt{30\,days}, \mathtt{NOW}]$. To be able to determine the exact effect of such an update, we must keep the fragment of the temporal base relation within the aggregate window.

3. ***Cumulative aggregates with single-period constraint.*** If the single-period constraint holds for R, it turns out that we only need the state of R^{nt} right before the start time of a source update in order to determine whether this update actually affects the value of a temporal aggregate. Therefore, if all aggregate functions in A are incrementally maintainable, we only need $\mathsf{FR}_{[\mathtt{NOW}-l-1,\mathtt{NOW}-1]}(R)$ as an auxiliary view. However, if A contains any aggregate function that is not incrementally maintainable, we still need $\mathsf{FR}_{[\mathtt{NOW}-l-l',\mathtt{NOW}]}(R)$ since the aggregate function may need to be recomputed.

Maintenance equations for temporal aggregates are given in [YW98b]. The auxiliary views specified for the various cases above guarantee that these maintenance equations can be evaluated without relying on historical data retrieved from the source. The auxiliary views themselves can be maintained using Algorithm 4.3. Hence, the entire set of views is self-maintainable.

7 Additional Issues and Scenarios

Temporal Projection. The temporal projection operator deserves special discussion because it is used in view definitions frequently and because it complicates the temporal view self-maintenance problem. There are two reasons why temporal projection makes a view difficult to maintain. The first reason stems from the set-based semantics of our data model. As discussed in Section 5, temporal projection views are snapshot-reducible, and hence are maintained in the same way as their nontemporal counterparts. In nontemporal relational algebra, a projection view is not self-maintainable with respect to deletions [QW91] unless the projection preserves the key of its operand. Consequently, a temporal projection view $\pi_A(R)$ is self-maintainable iff A contains a key of R. Note that if we change our data model from set semantics to bag semantics, as discussed in the next subsection, $\pi_A(R)$ would always be self-maintainable.

The second reason why temporal projection complicates self-maintenance is an effect called *coalescing* [BJS95], which is best illustrated by an example. Consider the temporal relation $S(\mathtt{Emp}, \mathtt{Dept}, \mathtt{T})$ shown in Figure 2. There are two tuples, s_1 and s_2, with the same `Emp` value but different `Dept` values. If we apply $\pi_{\{\mathtt{Emp}\}}$ to S, s_1 and s_2 will become one tuple whose `T` value is the result of coalescing $s_1.\mathtt{T}$ and $s_2.\mathtt{T}$, i.e., $(s_1.\mathtt{T} \cup s_2.\mathtt{T})$. After coalescing, the original `T` values cannot be recovered.

To see why coalescing complicates self-maintenance, consider a view of the form $\pi_A(\sigma_p(S))$, where p references `T` and A does not contain any key of S, e.g., $\pi_{\{\mathtt{Emp}\}}(\sigma_{\mathtt{length(T)<5\ days}}(S))$. It may seem reasonable to maintain $\pi_A(\sigma_{\alpha(p)}(S))$ as a superview, but unfortunately, this approach does not work because $\pi_A(\sigma_{\alpha(p)}(S))$ is not self-maintainable. When time advances, we need to re-evaluate the selection predicate in order to refresh the superview (Algorithm 3.1). In general, $\alpha(p)$ may refer to $S.\mathtt{T}$; however, if A contains no key of S, coalescing may occur, and we may lose the original `T` values of the S tuples. Another problem with using $\pi_A(\sigma_{\alpha(p)}(S))$ as a superview is that we cannot reconstruct the original view from it. Because of the coalescing effect, $\sigma_p\big(\pi_A(\sigma_{\alpha(p)}(S))\big) = \pi_A(\sigma_{p\wedge\alpha(p)}(S))$ does not always hold. In the general case, to make a temporal projection view self-maintainable, we must either augment the projection with a key of its operand, or store the entire operand as an auxiliary view.

Bag Semantics. We can change our data model from set semantics to bag (multiset) semantics with little effect on the rest of our framework. Under bag semantics, the value of time attribute `T` is a bag of time instants instead of a set. `T` also can be viewed as a set of time periods, each weighted by a positive count. A temporal bag R records the history of a nontemporal bag R^{nt}. A temporal tuple $r \in R$ is interpreted as follows: the nontemporal tuple $r.A$ appears in the state of R^{nt} at time instant t the same number of times as t appears in $r.\mathtt{T}$. Operators in temporal bag algebra are defined in the same way as operators in temporal relational algebra, except that bag operations are used on `T` instead of set operations. Each snapshot-reducible operator in temporal bag algebra corresponds to its counterpart in nontemporal bag algebra.

The algorithms of Sections 3 and 4 still work for temporal selection and join views under bag semantics, although we need to expand the tables in Appendix A to handle new predicates over a bag-valued `T`. For snapshot-reducible temporal bag views, we use the same algorithm of Section 5 to reduce the self-maintenance problem to its nontemporal version. However, the target of reduction is nontemporal bag algebra rather than relational algebra. Since projection views are self-maintainable in nontemporal bag algebra [GL95], temporal projection views also are self-maintainable under bag semantics. On the other hand, the effect of coalescing described above remains under bag semantics. Therefore, if the operand of a projection is not snapshot-reducible and hence requires view refresh, we still must ensure that the projection preserves the key of its operand, or we must store the entire operand as an auxiliary view.

Pure Source Deletions and Insertions. Recall that source updates ∇R and $\triangle R$ specify how to update `T` within the update window, but they may not provide the complete `T` values because data sources typically do not keep history outside their update windows. However, some sources may have a policy of keeping the complete history of all data that could potentially be updated. When updates occur, such sources can report to the warehouse complete old and new values for `T`, as a pair of "pure" deletions and insertions. Pure deletions always delete existing tuples completely, while pure insertions never expand `T` values of existing tuples, as described in Section 3.4. Properties of pure source deletions and insertions can be exploited to improve our selection superviews (Section 3). Since pure deletions and insertions do not modify `T`, α^{μ} becomes unnecessary, and α reduces to α^{t}. Other algorithms in this paper require no change, except that for a cumulative aggregate $\Pi_{A,l'}(R)$ (Section 6), no auxiliary views are necessary if all aggregate functions in A are incrementally maintainable.

Infinite Source Update Window. So far we have assumed that every data source has an update window $[\mathtt{NOW}-l, \mathtt{NOW}]$ for some constant l. What if a data source does not have any update-window constraint? Such sources may update arbitrary history retroactively at any point. Our framework handles such sources by assigning them the infinite update window $[\mathtt{NOW}-\infty, \mathtt{NOW}]$. Specifications of relaxed predicates, such as those in Appendix A, continue to be correct with l substituted by ∞, albeit in some cases they could be simplified further. Our algorithms for deriving and maintaining auxiliary views continue to work as well. FR_w becomes an identity operation for $w = [\mathtt{NOW}-\infty, \mathtt{NOW}]$; in other words, we cannot discard any history if the update window is infinite.

8 Related and Future Work

Numerous temporal data models and query languages have been proposed; see [ÖS95] for a survey. We have chosen BCDM [JSS94], the underlying data model for TSQL2 [BJS95], as a basis for our work. Moving-window aggregates were discussed in [NA89, Sno87]. As future work, we will investigate how to extend our framework to handle other temporal models and languages.

The concept of snapshot-reducibility was introduced in [Sno87, BJS95], and its relevance to temporal view maintenance was pointed out in [YW98a].

The view maintenance problem has received increasing attention recently [GM95] because of its application in data warehousing [Wid95]. However, most work considers only nontemporal views. In [YW98a], we studied incremental maintenance of temporal views in a data warehousing environment, laying the foundation for our work in this paper. Both [YW98a] and this paper derive change propagation equations for snapshot-reducible temporal views from related work in nontemporal view maintenance [QW91, GL95, Qua96].

The notion of view self-maintainability was introduced in [BCL89, GJM96], where they provided tests for determining whether a nontemporal view is self-maintainable. [HZ96, QGMW96, AJB98] studied the problem of making a nontemporal view self-maintainable by storing auxiliary views. This paper addresses the corresponding problem for temporal views. As discussed in Section 5, the self-maintenance problem for snapshot-reducible temporal views can be reduced to the nontemporal version. However, temporal views that are not snapshot-reducible require novel techniques presented in Sections 3, 4, and 6.

A number of papers are related to the temporal view self-maintenance problem to some degree. [JMS95] explored view maintenance in the *chronicle* model. The chronicle model makes several assumptions that simplify view self-maintenance. First, the model does not support `NOW`. Each tuple in a chronicle is timestamped with a single time instant rather than a set. Operators of the chronicle algebra are essentially snapshot-reducible. Finally, chronicles are append-only and no retroactive updates are allowed; using our terminology, the update window is [`NOW`, `NOW`]. It is possible to handle the chronicle model as a special case in our framework.

[BM95] considered the maintenance of relational queries whose selection predicate may reference `NOW`. They also used the idea of superviews to make selection views self-maintainable. Under their data model, updates to the time attribute are modeled as pure deletions and insertions. Therefore, as discussed in Section 7, it suffices to consider only the dimension of time advances. Moreover, they assumed that `NOW` could not be stored in data, and there were no update-window constraints. These assumptions further simplify the problem in their case.

[GMLY98] developed a framework for system-managed expiration of warehouse data: the warehouse administrator can declare constraints that apply to warehouse data as well as source updates, and the system uses this knowledge to identify and expire data that is not needed for view maintenance. Although intended for nontemporal views, this framework also can be used to expire historical data that is not needed for maintaining some simple temporal views with update constraints, such as the chronicles discussed above [JMS95]. However, this framework cannot handle the general case of temporal views whose maintenance may require view refresh.

[Cho95] considered the problem of monitoring temporal integrity constraints by storing auxiliary data containing enough historical information to check constraints, so that the entire database history need not be stored. Their language is logical rather than algebraic, and is generally less expressive than ours. Retroactive updates were not considered. Furthermore, the problem of detecting constraint violations is a strict subset of the problem of maintaining views.

In addition to the future work mentioned before, we plan to extend our current temporal view definition language to include the fragment operator FR from Section 4, which allows views containing partial history to be specified. Another interesting extension is to consider how to make a set of temporal views self-maintainable so that data in the auxiliary views can be shared.

References

[AJB98] M. O. Akinde, O. G. Jensen, and M. H. Böhlen. Minimizing detail data in data warehouses. In *Proc. of the 1998 Intl. Conf. on Extending Database Technology*, pages 293–307, March 1998.

[BCL89] J. A. Blakeley, N. Coburn, and P.-Å. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. on Database Systems*, 14(3):369–400, September 1989.

[BJS95] M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Evaluating the completeness of TSQL2. In *Proc. of the 1995 Intl. Workshop on Temporal Databases*, pages 153–174, September 1995.

[BM95] L. Bækgaard and L. Mark. Incremental computation of time-varying query expressions. *IEEE Trans. on Knowledge and Data Engineering*, 7(4):583–590, August 1995.

[CDI+97] J. Clifford, C. E. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the semantics of "NOW" in databases. *ACM Trans. on Database Systems*, 22(2):171–214, June 1997.

[Cho95] J. Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. on Database Systems*, 20(2):148–186, June 1995.

[GJM96] A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views. In *Proc. of the 1996 Intl. Conf. on Extending Database Technology*, pages 140–144, March 1996.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. of the 1995 ACM SIGMOD Intl. Conf. on Management of Data*, pages 328–339, May 1995.

[GM95] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, June 1995.

[GMLY98] H. Garcia-Molina, W. J. Labio, and J. Yang. Expiring data in a warehouse. In *Proc. of the 1998 Intl. Conf. on Very Large Data Bases*, pages 500–511, August 1998.

[HZ96] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *Proc. of the 1996 ACM SIGMOD Intl. Conf. on Management of Data*, pages 481–492, June 1996.

[JMS95] H. V. Jagadish, I. S. Mumick, and A. Silberschatz. View maintenance issues for the chronicle data model. In *Proc. of the 1995 ACM Symp. on Principles of Database Systems*, pages 113–124, May 1995.

[JSS94] C. S. Jensen, M. D. Soo, and R. T. Snodgrass. Unifying temporal models via a conceptual model. *Information Systems*, 19(7):513–547, 1994.

[NA89] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49(1):147–175, 1989.

[ÖS95] G. Özsoyoğlu and R. Snodgrass. Temporal and real-time databases: A survey. *IEEE Trans. on Knowledge and Data Engineering*, 7(4):513–532, August 1995.

[QGMW96] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Proc. of the 1996 Intl. Conf. on Parallel and Distributed Information Systems*, pages 158–169, December 1996.

[Qua96] D. Quass. Maintenance expressions for views with aggregation. In *Proc. of the 1996 ACM Workshop on Materialized Views: Techniques and Applications*, pages 110–118, June 1996.

[QW91] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Trans. on Knowledge and Data Engineering*, 3(3):337–341, September 1991.

[Sno87] R. T. Snodgrass. The temporal query language TQuel. *ACM Trans. on Database Systems*, 12(2):247–298, June 1987.

[Wid95] J. Widom. Research problems in data warehousing. In *Proc. of the 1995 Intl. Conf. on Information and Knowledge Management*, pages 25–30, November 1995.

[YW98a] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proc. of the 1998 Intl. Conf. on Extending Database Technology*, pages 389–403, March 1998.

[YW98b] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. Technical report, Computer Science Department, Stanford University, January 1998.
`http://www-db.stanford.edu/pub/papers/yw-tempview.ps`.

A Specification of Relaxed Predicates

The following grammar describes the atomic predicates considered in our work:

$$
\begin{array}{rcl}
\textit{comp} & \rightarrow & = \ | \ \neq \ | \ > \ | \ \geq \ | \ < \ | \ \leq \\
\textit{time} & \rightarrow & t \ | \ \texttt{NOW} - k \\
\textit{period} & \rightarrow & [\textit{time}_1, \textit{time}_2] \\
p & \rightarrow & \text{a nontemporal predicate with no references to } \texttt{T} \text{ or } \texttt{NOW} \\
& | & \textit{time}_1 \ \textit{comp} \ \textit{time}_2 \\
& | & (\texttt{TS} \mid \texttt{TE}) \ \textit{comp} \ \textit{time} \\
& | & (\texttt{TS} \mid \texttt{TE}) \ \texttt{is} \ [\texttt{not}] \ \texttt{NOW} \\
& | & \texttt{length}(\texttt{T}) \ \textit{comp} \ k \\
& | & \texttt{T} \ [\texttt{not}] \ \texttt{overlaps} \ \textit{period} \\
& | & \texttt{T} \ [\texttt{not}] \ \texttt{contains} \ \textit{period} \\
& | & \textit{period} \ [\texttt{not}] \ \texttt{contains} \ \texttt{T}
\end{array}
$$

In the above, t is a time instant contant and k is a nonnegative length. This set of atomic predicates is closed under negation. The meanings of the predicates are explained in Section 2.1. A period $d = [\textit{time}_1, \textit{time}_2]$ is empty when $\textit{time}_1 > \textit{time}_2$. For this special case, ($\texttt{T not overlaps } d$), ($\texttt{T contains } d$), and ($d \texttt{ not contains T}$) are true, while ($\texttt{T overlaps } d$), ($\texttt{T not contains } d$), and ($d \texttt{ contains T}$) are false. Note that whether d is empty or not may depend on the current time since either $\textit{time}_1$ or $\textit{time}_2$ may be NOW-relative.

Tables 1–6 specify the relaxed predicates for all atomic predicates. For all tables, unless otherwise noted, we assume $t_1 \leq t_2$, $k \geq 0$, $k_1 \geq k_2 \geq 0$, and the update window is $[\texttt{NOW} - l, \texttt{NOW}]$, where $l \geq 0$.

To simplify the tables, we omit atomic predicates that are either always true or always false, because their corresponding relaxed predicates are simply `true` or `false`, respectively. For example, ($\texttt{NOW} - k_1 > \texttt{NOW} - k_2$) is omitted because it is either always true or always false depending the values of k_1 and k_2, which are fixed. We also omit predicates that can be expressed easily using other predicates listed in the tables. For example, ($\texttt{TE} \neq t$) can be expressed as $(\texttt{TE} < t) \vee (\texttt{TE} > t)$; ($[t_1, t_2] \texttt{ not contains T}$) can be expressed as $(\texttt{TS} < t_1) \vee (\texttt{TE} > t_2)$.

Tables 2, 4, and 6 assume that `T` contains only a single period. For these tables, we do not list `overlaps` and `contains` predicates because they can be expressed using `TS` and `TE` comparisons. For example, ($\texttt{T contains } [t_1, t_2]$) can be expressed as $(\texttt{TS} \leq t_1) \wedge (\texttt{TE} \geq t_2)$. On the other hand, without the single-period constraint, `overlaps` and `contains` are not expressible using `TS` and `TE` comparisons since there could be gaps between the periods in `T`. Therefore, `overlaps` and `contains` are still listed in Tables 1, 3, and 5.

In Row 28 of Table 1, p_{28} denotes the predicate: "there is a gap of length greater than $k_1 - k_2$ in the part of `T` after $\texttt{NOW} - k_1$." In Row 35 of Table 1, p_{35} denotes the predicate: "there is a period of length greater than $k_1 - k_2$ in the part of `T` after $\texttt{NOW} - k_1$." Although p_{28} and p_{35} are not expressible in our predicate language, they are supported by typical temporal query engines such as TSQL2 [BJS95]. If not supported, we can relax them further, in the worst case to `true`, but the minimality of the specification will be lost.

	p	$\hat{\alpha}^t(p)$
1	p is nontemporal	same as p
2	$\texttt{NOW} = t$	$\texttt{NOW} \leq t$
3	$\texttt{NOW} > t$	$\texttt{true}$
4	$\texttt{NOW} < t$	$\texttt{NOW} < t$
5	$\texttt{TS} = t$	$\texttt{TS} = t$
6	$\texttt{TS} > t$	$\texttt{TS} > t$
7	$\texttt{TS} < t$	$\texttt{TS} < t$
8	$\texttt{TS} = \texttt{NOW} - k$	$\texttt{TS} \geq \texttt{NOW} - k$
9	$\texttt{TS} > \texttt{NOW} - k$	$\texttt{TS} > \texttt{NOW} - k$
10	$\texttt{TS} < \texttt{NOW} - k$	$\texttt{true}$
11	$\texttt{TE} = t$	$\texttt{TE} = t \vee (\texttt{TE is NOW} \wedge \texttt{NOW} < t)$
12	$\texttt{TE} > t$	$\texttt{TE} > t \vee (\texttt{TE is NOW} \wedge \texttt{NOW} \leq t)$
13	$\texttt{TE} < t$	$\texttt{TE} < t$
14	$\texttt{TE is NOW}$	$\texttt{TE is NOW}$
15	$\texttt{TE is not NOW}$	$\texttt{TE is not NOW}$
16	$\texttt{TE} = \texttt{NOW} - k \ (k > 0)$	$\texttt{TE} \geq \texttt{NOW} - k \wedge \texttt{TE is not NOW}$
17	$\texttt{TE} = \texttt{NOW} - k \ (k = 0)$	see Row 14
18	$\texttt{TE} > \texttt{NOW} - k$	$\texttt{TE} > \texttt{NOW} - k$
19	$\texttt{TE} < \texttt{NOW} - k$	$\texttt{TE is not NOW}$
20	$\texttt{length(T)} = k \ (k \geq 1)$	$\texttt{length(T)} = k \vee (\texttt{length(T)} < k \wedge \texttt{TE is NOW})$
21	$\texttt{length(T)} = k \ (k = 0)$	$\texttt{false}$
22	$\texttt{length(T)} > k$	$\texttt{length(T)} > k \vee \texttt{TE is NOW}$
23	$\texttt{length(T)} < k \ (k \geq 2)$	$\texttt{length(T)} < k$
24	$\texttt{length(T)} < k \ (k = 0, 1)$	$\texttt{false}$
25	$\texttt{T overlaps } [t_1, t_2]$	$\texttt{T overlaps } [t_1, t_2] \vee (\texttt{TE is NOW} \wedge \texttt{NOW} < t_1)$
26	$\texttt{T not overlaps } [t_1, t_2]$	$\texttt{T not overlaps } [t_1, t_2]$
27	$\texttt{T overlaps } [\texttt{NOW} - k_1, \texttt{NOW} - k_2]$	$\texttt{T overlaps } [\texttt{NOW} - k_1, \texttt{NOW} - k_2] \vee \texttt{TE} > \texttt{NOW} - k_2$
†28	$\texttt{T not overlaps } [\texttt{NOW} - k_1, \texttt{NOW} - k_2]$	$\texttt{T not overlaps } [\texttt{NOW} - k_1, \texttt{NOW} - k_2] \vee \texttt{TE is not NOW} \vee p_{28}$
29	$\texttt{T overlaps } [t, \texttt{NOW} - k]$	$\texttt{T overlaps } [t, \texttt{NOW} - k] \vee \texttt{TE is NOW} \vee (\texttt{TE} > t \wedge \texttt{TE} > \texttt{NOW} - k)$
30	$\texttt{T not overlaps } [t, \texttt{NOW} - k]$	$\texttt{T not overlaps } [t, \texttt{NOW} - k]$
31	$\texttt{T overlaps } [\texttt{NOW} - k, t]$	$\texttt{T overlaps } [\texttt{NOW} - k, t]$
32	$\texttt{T not overlaps } [\texttt{NOW} - k, t]$	$\texttt{true}$
33	$\texttt{T contains } [t_1, t_2]$	$\texttt{T contains } [t_1, t_2] \vee (\texttt{TE is NOW} \wedge \texttt{T contains } [t_1, \texttt{NOW}])$
34	$\texttt{T not contains } [t_1, t_2]$	$\texttt{T not contains } [t_1, t_2]$
†35	$\texttt{T contains } [\texttt{NOW} - k_1, \texttt{NOW} - k_2]$	$\texttt{T contains } [\texttt{NOW} - k_1, \texttt{NOW} - k_2] \vee \texttt{TE is NOW} \vee p_{35}$
36	$\texttt{T not contains } [\texttt{NOW} - k_1, \texttt{NOW} - k_2]$	$\texttt{T not contains } [\texttt{NOW} - k_1, \texttt{NOW}]$
37	$\texttt{T contains } [t, \texttt{NOW} - k]$	$\texttt{T contains } [t, \texttt{NOW} - k]$
38	$\texttt{T not contains } [t, \texttt{NOW} - k]$	$\texttt{T not contains } [t, \texttt{NOW}] \vee (\texttt{NOW} < t \wedge \texttt{TE is not NOW})$
39	$\texttt{T contains } [\texttt{NOW} - k, t]$	$\texttt{true}$
40	$\texttt{T not contains } [\texttt{NOW} - k, t]$	$\texttt{T not contains } [\texttt{NOW} - k, t]$

† See Appendix A text.

Table 1: $\hat{\alpha}^t$; $\texttt{T}$ can be a set of periods.

	p	$\hat{\alpha}^t(p)$
1	p is nontemporal	same as p
2	$\texttt{NOW} = t$	$\texttt{NOW} \leq t$
3	$\texttt{NOW} > t$	$\texttt{true}$
4	$\texttt{NOW} < t$	$\texttt{NOW} < t$
5	$\texttt{TS} = t$	$\texttt{TS} = t$
6	$\texttt{TS} > t$	$\texttt{TS} > t$
7	$\texttt{TS} < t$	$\texttt{TS} < t$
8	$\texttt{TS} = \texttt{NOW} - k$	$\texttt{TS} \geq \texttt{NOW} - k$
9	$\texttt{TS} > \texttt{NOW} - k$	$\texttt{TS} > \texttt{NOW} - k$
10	$\texttt{TS} < \texttt{NOW} - k$	$\texttt{true}$
11	$\texttt{TE} = t$	$\texttt{TE} = t \vee (\texttt{TE is NOW} \wedge \texttt{NOW} < t)$
12	$\texttt{TE} > t$	$\texttt{TE} > t \vee (\texttt{TE is NOW} \wedge \texttt{NOW} \leq t)$
13	$\texttt{TE} < t$	$\texttt{TE} < t$
14	$\texttt{TE is NOW}$	$\texttt{TE is NOW}$
15	$\texttt{TE is not NOW}$	$\texttt{TE is not NOW}$
16	$\texttt{TE} = \texttt{NOW} - k$ $(k > 0)$	$\texttt{TE} \geq \texttt{NOW} - k \wedge \texttt{TE is not NOW}$
17	$\texttt{TE} = \texttt{NOW} - k$ $(k = 0)$	see Row 14
18	$\texttt{TE} > \texttt{NOW} - k$	$\texttt{TE} > \texttt{NOW} - k$
19	$\texttt{TE} < \texttt{NOW} - k$	$\texttt{TE is not NOW}$
20	$\texttt{length}(\texttt{T}) = k$ $(k \geq 1)$	$\texttt{length}(\texttt{T}) = k \vee (\texttt{length}(\texttt{T}) < k \wedge \texttt{TE is NOW})$
21	$\texttt{length}(\texttt{T}) = k$ $(k = 0)$	$\texttt{false}$
22	$\texttt{length}(\texttt{T}) > k$	$\texttt{length}(\texttt{T}) > k \vee \texttt{TE is NOW}$
23	$\texttt{length}(\texttt{T}) < k$ $(k \geq 2)$	$\texttt{length}(\texttt{T}) < k$
24	$\texttt{length}(\texttt{T}) < k$ $(k = 0, 1)$	$\texttt{false}$

Table 2: $\hat{\alpha}^t$; $\texttt{T}$ contains a single period.

	p	$\hat{\alpha}^{\mu}(p)$
1	p is nontemporal	same as p
2	$\texttt{NOW} = t$	$\texttt{NOW} = t$
3	$\texttt{NOW} > t$	$\texttt{NOW} > t$
4	$\texttt{NOW} < t$	$\texttt{NOW} < t$
5	$\texttt{TS} = t$	$(\texttt{NOW} - l > t \wedge \texttt{TS} = t) \vee (\texttt{NOW} - l \leq t \wedge \texttt{NOW} \geq t \wedge \texttt{TS} \geq \texttt{NOW} - l)$
6	$\texttt{TS} > t$	$(\texttt{NOW} - l > t \wedge \texttt{TS} > t) \vee (\texttt{NOW} - l \leq t \wedge \texttt{NOW} > t \wedge \texttt{TS} \geq \texttt{NOW} - l)$
7	$\texttt{TS} < t$	$(\texttt{NOW} - l \geq t \wedge \texttt{TS} < t) \vee \texttt{NOW} - l < t$
8	$\texttt{TS} = \texttt{NOW} - k\ (k > l)$	$\texttt{TS} = \texttt{NOW} - k$
9	$\texttt{TS} = \texttt{NOW} - k\ (k \leq l)$	$\texttt{TS} \geq \texttt{NOW} - l$
10	$\texttt{TS} > \texttt{NOW} - k\ (k > l)$	$\texttt{TS} > \texttt{NOW} - k$
11	$\texttt{TS} > \texttt{NOW} - k\ (0 < k \leq l)$	$\texttt{TS} \geq \texttt{NOW} - l$
12	$\texttt{TS} > \texttt{NOW} - k\ (k = 0)$	$\texttt{false}$
13	$\texttt{TS} < \texttt{NOW} - k\ (k \geq l)$	$\texttt{TS} < \texttt{NOW} - k$
14	$\texttt{TS} < \texttt{NOW} - k\ (k < l)$	$\texttt{true}$
15	$\texttt{TE} = t$	$(\texttt{NOW} - l > t \wedge \texttt{T not overlaps } [t+1, \texttt{NOW} - l - 1] \wedge \texttt{T overlaps } [t,t])$ $\vee (\texttt{NOW} - l \leq t \wedge \texttt{NOW} \geq t)$
16	$\texttt{TE} > t$	$\texttt{NOW} > t$
17	$\texttt{TE} < t$	$(\texttt{NOW} - l \geq t \wedge \texttt{T not overlaps } [t, \texttt{NOW} - l - 1] \wedge \texttt{TS} < t) \vee \texttt{NOW} - l < t$
18	$\texttt{TE is NOW}$	$\texttt{true}$
19	$\texttt{TE is not NOW}\ (l > 0)$	$\texttt{true}$
20	$\texttt{TE is not NOW}\ (l = 0)$	$\texttt{TS} < \texttt{NOW}$
21	$\texttt{TE} = \texttt{NOW} - k\ (k > l)$	$\texttt{T not overlaps } [\texttt{NOW} - k + 1, \texttt{NOW} - l - 1] \wedge \texttt{T overlaps } [\texttt{NOW} - k, \texttt{NOW} - k]$
22	$\texttt{TE} = \texttt{NOW} - k\ (k \leq l)$	$\texttt{true}$
23	$\texttt{TE} > \texttt{NOW} - k\ (k > 0)$	$\texttt{true}$
24	$\texttt{TE} > \texttt{NOW} - k\ (k = 0)$	$\texttt{false}$
25	$\texttt{TE} < \texttt{NOW} - k\ (k \geq l)$	$\texttt{T not overlaps } [\texttt{NOW} - k, \texttt{NOW} - l - 1] \wedge \texttt{TS} < \texttt{NOW} - k$
26	$\texttt{TE} < \texttt{NOW} - k\ (k < l)$	$\texttt{true}$
27	$\texttt{length(T)} = k\ (k \geq 1)$	$k - l - 1 \leq \texttt{length(T} - [\texttt{NOW} - l, \texttt{NOW}]) \leq k$
28	$\texttt{length(T)} = k\ (k = 0)$	$\texttt{false}$
29	$\texttt{length(T)} > k$	$\texttt{length(T} - [\texttt{NOW} - l, \texttt{NOW}]) > k - l - 1$
30	$\texttt{length(T)} < k\ (k \geq 2)$	$\texttt{length(T} - [\texttt{NOW} - l, \texttt{NOW}]) < k$
31	$\texttt{length(T)} < k\ (k = 0, 1)$	$\texttt{false}$

Table 3: $\hat{\alpha}^{\mu}$; $\texttt{T}$ can be a set of periods. (Continued on next page.)

	p	$\hat{\alpha}^{\mu}(p)$
32	T overlaps $[t_1, t_2]$	T overlaps $[t_1, t_2] \vee [t_1, t_2]$ overlaps $[\mathtt{NOW} - l, \mathtt{NOW}]$
33	T not overlaps $[t_1, t_2]$	T not overlaps $[t_1, t_2] - [\mathtt{NOW} - l, \infty]$ $\wedge\, (\mathtt{NOW} > t_2 \vee \mathtt{NOW} - l < t_1 \vee \mathtt{TS} < t_1)$
34	T overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 > l)$	T overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$
35	T overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 \leq l)$	**true**
36	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 > l)$	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$
37	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(0 < k_2 \leq l < k_1)$	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - l - 1]$
38	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(0 < k_2 \leq k_1 \leq l)$	**true**
39	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 = 0)$	see Rows 25 and 26
40	T overlaps $[t, \mathtt{NOW} - k]$ $(k > l)$	T overlaps $[t, \mathtt{NOW} - k]$
41	T overlaps $[t, \mathtt{NOW} - k]$ $(k \leq l)$	$\mathtt{NOW} - k \geq t$
42	T not overlaps $[t, \mathtt{NOW} - k]$ $(k > l)$	T not overlaps $[t, \mathtt{NOW} - k]$
43	T not overlaps $[t, \mathtt{NOW} - k]$ $(0 < k \leq l)$	T not overlaps $[t, \mathtt{NOW} - l - 1]$
44	T not overlaps $[t, \mathtt{NOW} - k]$ $(k = 0)$	see Row 17
45	T overlaps $[\mathtt{NOW} - k, t]$ $(k > l)$	T overlaps $[\mathtt{NOW} - k, t] \vee \mathtt{NOW} - l \leq t$
46	T overlaps $[\mathtt{NOW} - k, t]$ $(k \leq l)$	$\mathtt{NOW} - k \leq t$
47	T not overlaps $[\mathtt{NOW} - k, t]$ $(k > l)$	(T not overlaps $[\mathtt{NOW} - k, t] \wedge \mathtt{NOW} - l > t)$ $\vee$ (T not overlaps $[\mathtt{NOW} - k, \mathtt{NOW} - l - 1]$ $\wedge\, \mathtt{NOW} - l \leq t \wedge \mathtt{NOW} > t)$ $\vee$ (T not overlaps $[\mathtt{NOW} - k, \mathtt{NOW} - l - 1]$ $\wedge\, \mathtt{NOW} - l \leq t \wedge \mathtt{TS} < \mathtt{NOW} - k)$
48	T not overlaps $[\mathtt{NOW} - k, t]$ $(k = l)$	$\mathtt{NOW} > t \vee \mathtt{TS} < \mathtt{NOW} - k$
49	T not overlaps $[\mathtt{NOW} - k, t]$ $(k < l)$	**true**
50	T contains $[t_1, t_2]$	T contains $[t_1, t_2] \vee (\mathtt{NOW} - l \leq t_2 \wedge \mathtt{NOW} \geq t_2$ $\wedge$ T contains $[t_1, \mathtt{NOW} - l - 1])$
51	T not contains $[t_1, t_2]$ $(l > 0)$	T not contains $[t_1, t_2]$ $\vee\, [t_1, t_2]$ overlaps $[\mathtt{NOW} - l, \mathtt{NOW}]$
52	T not contains $[t_1, t_2]$ $(l = 0)$	T not contains $[t_1, t_2]$ $\vee\, (\mathtt{NOW} \geq t_1 \wedge \mathtt{NOW} \leq t_2 \wedge \mathtt{TS} < \mathtt{NOW})$
53	T contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 > l)$	T contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$
54	T contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 \leq l)$	T contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - l - 1]$
55	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 > l)$	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$
56	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 \leq l,\ l > 0)$	**true**
57	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$ $(k_2 = l = 0)$	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2] \vee \mathtt{TS} < \mathtt{NOW}$
58	T contains $[t, \mathtt{NOW} - k]$ $(k > l)$	T contains $[t, \mathtt{NOW} - k]$
59	T contains $[t, \mathtt{NOW} - k]$ $(k \leq l)$	T contains $[t, \mathtt{NOW} - l - 1]$
60	T not contains $[t, \mathtt{NOW} - k]$ $(k > l)$	T not contains $[t, \mathtt{NOW} - k]$
61	T not contains $[t, \mathtt{NOW} - k]$ $(k \leq l,\ l > 0)$	$\mathtt{NOW} - k \geq t$
62	T not contains $[t, \mathtt{NOW} - k]$ $(k = l = 0)$	T not contains $[t, \mathtt{NOW}] \vee (\mathtt{NOW} \geq t \wedge \mathtt{TS} < \mathtt{NOW})$
63	T contains $[\mathtt{NOW} - k, t]$ $(k > l)$	$(\mathtt{NOW} - l - 1 \leq t$ $\wedge$ T contains $[\mathtt{NOW} - k, \mathtt{NOW} - l - 1])$ $\vee\, (\mathtt{NOW} - l - 1 > t$ $\wedge$ T contains $[\mathtt{NOW} - k, \mathtt{NOW} - t])$
64	T contains $[\mathtt{NOW} - k, t]$ $(k \leq l)$	$\mathtt{NOW} \geq t$
65	T not contains $[\mathtt{NOW} - k, t]$ $(k > l > 0)$	T not contains $[\mathtt{NOW} - k, t] \vee \mathtt{NOW} - l \leq t$
66	T not contains $[\mathtt{NOW} - k, t]$ $(k \leq l, l > 0)$	$\mathtt{NOW} - k \leq t$
67	T not contains $[\mathtt{NOW} - k, t]$ $(l = 0)$	T not contains $[\mathtt{NOW} - k, t] \vee (\mathtt{NOW} \leq t \wedge \mathtt{TS} < \mathtt{NOW})$

Table 3: $\hat{\alpha}^{\mu}$; T can be a set of periods. (Continued from previous page.)

	p	$\hat{\alpha}^{\mu}(p)$
1	p is nontemporal	same as p
2	$\mathtt{NOW} = t$	$\mathtt{NOW} \leq t$
3	$\mathtt{NOW} > t$	$\mathtt{true}$
4	$\mathtt{NOW} < t$	$\mathtt{NOW} < t$
5	$\mathtt{TS} = t$	$(\mathtt{NOW} - l > t \wedge \mathtt{TS} = t) \vee (\mathtt{NOW} - l \leq t \wedge \mathtt{NOW} \geq t \wedge \mathtt{TS} \geq \mathtt{NOW} - l)$
6	$\mathtt{TS} > t$	$(\mathtt{NOW} - l > t \wedge \mathtt{TS} > t) \vee (\mathtt{NOW} - l \leq t \wedge \mathtt{NOW} > t \wedge \mathtt{TS} \geq \mathtt{NOW} - l)$
7	$\mathtt{TS} < t$	$(\mathtt{NOW} - l \geq t \wedge \mathtt{TS} < t) \vee \mathtt{NOW} - l < t$
8	$\mathtt{TS} = \mathtt{NOW} - k \ (k > l)$	$\mathtt{TS} = \mathtt{NOW} - k$
9	$\mathtt{TS} = \mathtt{NOW} - k \ (k \leq l)$	$\mathtt{TS} \geq \mathtt{NOW} - l$
10	$\mathtt{TS} > \mathtt{NOW} - k \ (k > l)$	$\mathtt{TS} > \mathtt{NOW} - k$
11	$\mathtt{TS} > \mathtt{NOW} - k \ (0 < k \leq l)$	$\mathtt{TS} \geq \mathtt{NOW} - l$
12	$\mathtt{TS} > \mathtt{NOW} - k \ (k = 0)$	$\mathtt{false}$
13	$\mathtt{TS} < \mathtt{NOW} - k \ (k \geq l)$	$\mathtt{TS} < \mathtt{NOW} - k$
14	$\mathtt{TS} < \mathtt{NOW} - k \ (k < l)$	$\mathtt{true}$
15	$\mathtt{TE} = t$	$(\mathtt{NOW} - l - 1 > t \wedge \mathtt{TE} = t)$ $\vee\, (\mathtt{NOW} - l - 1 = t \wedge \mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \leq \mathtt{NOW} - l - 1)$ $\vee\, (\mathtt{NOW} - l \leq t \wedge \mathtt{NOW} \geq t \wedge \mathtt{TE} \geq \mathtt{NOW} - l - 1)$
16	$\mathtt{TE} > t$	$\mathtt{TE} > t \vee (\mathtt{NOW} > t \wedge \mathtt{TE} \geq \mathtt{NOW} - l - 1)$
17	$\mathtt{TE} < t$	$(\mathtt{NOW} - l > t \wedge \mathtt{TE} < t) \vee (\mathtt{NOW} - l = t \wedge \mathtt{TS} < t) \vee \mathtt{NOW} - l < t$
18	$\mathtt{TE}$ $\mathtt{is}$ $\mathtt{NOW}$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
19	$\mathtt{TE}$ $\mathtt{is}$ $\mathtt{not}$ $\mathtt{NOW}$ $(l > 0)$	$\mathtt{true}$
20	$\mathtt{TE}$ $\mathtt{is}$ $\mathtt{not}$ $\mathtt{NOW}$ $(l = 0)$	$\mathtt{TS} < \mathtt{NOW}$
21	$\mathtt{TE} = \mathtt{NOW} - k \ (k > l + 1)$	$\mathtt{TE} = \mathtt{NOW} - k$
22	$\mathtt{TE} = \mathtt{NOW} - k \ (k = l + 1)$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \leq \mathtt{NOW} - l - 1$
23	$\mathtt{TE} = \mathtt{NOW} - k \ (k \leq l)$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
24	$\mathtt{TE} > \mathtt{NOW} - k \ (k > l + 1)$	$\mathtt{TE} > \mathtt{NOW} - k$
25	$\mathtt{TE} > \mathtt{NOW} - k \ (0 < k \leq l + 1)$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
26	$\mathtt{TE} > \mathtt{NOW} - k \ (k = 0)$	$\mathtt{false}$
27	$\mathtt{TE} < \mathtt{NOW} - k \ (k > l)$	$\mathtt{TE} < \mathtt{NOW} - k$
28	$\mathtt{TE} < \mathtt{NOW} - k \ (k = l)$	$\mathtt{TS} < \mathtt{NOW} - l$
29	$\mathtt{TE} < \mathtt{NOW} - k \ (k < l)$	$\mathtt{true}$
30	$\mathtt{length}(\mathtt{T}) = k \ (k > l + 1)$	$\mathtt{length}(\mathtt{T}) = k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \geq \mathtt{NOW} - l - k \wedge \mathtt{TS} \leq \mathtt{NOW} - k + 1)$
31	$\mathtt{length}(\mathtt{T}) = k \ (1 \leq k \leq l + 1)$	$\mathtt{length}(\mathtt{T}) = k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \geq \mathtt{NOW} - l - k)$
32	$\mathtt{length}(\mathtt{T}) = k \ (k = 0)$	$\mathtt{false}$
33	$\mathtt{length}(\mathtt{T}) > k \ (k > l)$	$\mathtt{length}(\mathtt{T}) > k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} < \mathtt{NOW} - k + 1)$
34	$\mathtt{length}(\mathtt{T}) > k \ (k \leq l)$	$\mathtt{length}(\mathtt{T}) > k \vee \mathtt{TE} \geq \mathtt{NOW} - l - 1$
35	$\mathtt{length}(\mathtt{T}) < k \ (k \geq 2)$	$\mathtt{length}(\mathtt{T}) < k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} > \mathtt{NOW} - l - k)$
36	$\mathtt{length}(\mathtt{T}) < k \ (k = 0, 1)$	$\mathtt{false}$

Table 4: $\hat{\alpha}^{\mu}$; $\mathtt{T}$ contains a single period.

	p	$\hat{\alpha}(p)$
1	p is nontemporal	same as p
2	$\texttt{NOW} = t$	$\texttt{NOW} \leq t$
3	$\texttt{NOW} > t$	$\texttt{true}$
4	$\texttt{NOW} < t$	$\texttt{NOW} < t$
5	$\texttt{TS} = t \ (l > 0)$	$\texttt{TS} = t \vee (\texttt{NOW} - l \leq t \wedge \texttt{TS} \geq \texttt{NOW} - l)$
6	$\texttt{TS} = t \ (l = 0)$	$\texttt{TS} = t$
7	$\texttt{TS} > t \ (l > 0)$	$\texttt{TS} > t \vee (\texttt{NOW} - l \leq t \wedge \texttt{TS} \geq \texttt{NOW} - l)$
8	$\texttt{TS} > t \ (l = 0)$	$\texttt{TS} > t$
9	$\texttt{TS} < t$	$\texttt{TS} < t \vee \texttt{NOW} - l < t$
10	$\texttt{TS} = \texttt{NOW} - k \ (k > l)$	$\texttt{TS} \geq \texttt{NOW} - k$
11	$\texttt{TS} = \texttt{NOW} - k \ (k \leq l)$	$\texttt{TS} \geq \texttt{NOW} - l$
12	$\texttt{TS} > \texttt{NOW} - k \ (k > l)$	$\texttt{TS} > \texttt{NOW} - k$
13	$\texttt{TS} > \texttt{NOW} - k \ (0 < k \leq l)$	$\texttt{TS} \geq \texttt{NOW} - l$
14	$\texttt{TS} > \texttt{NOW} - k \ (k = 0)$	$\texttt{false}$
15	$\texttt{TS} < \texttt{NOW} - k$	$\texttt{true}$
16	$\texttt{TE} = t$	$(\texttt{NOW} - l > t \wedge \texttt{T not overlaps } [t+1, \texttt{NOW} - l - 1] \wedge \texttt{T overlaps } [t, t])$ $\vee\ \texttt{NOW} - l \leq t$
17	$\texttt{TE} > t$	$\texttt{true}$
18	$\texttt{TE} < t$	$(\texttt{NOW} - l \geq t \wedge \texttt{T not overlaps } [t, \texttt{NOW} - l - 1] \wedge \texttt{TS} < t) \vee \texttt{NOW} - l < t$
19	$\texttt{TE is NOW}$	$\texttt{true}$
20	$\texttt{TE is not NOW}$	$\texttt{true}$
21	$\texttt{TE} = \texttt{NOW} - k$	$\texttt{true}$
22	$\texttt{TE} > \texttt{NOW} - k \ (k > 0)$	$\texttt{true}$
23	$\texttt{TE} > \texttt{NOW} - k \ (k = 0)$	$\texttt{false}$
24	$\texttt{TE} < \texttt{NOW} - k$	$\texttt{true}$
25	$\texttt{length(T)} = k \ (k \geq 1)$	$\texttt{length(T} - [\texttt{NOW} - l, \texttt{NOW}]) \leq k$
26	$\texttt{length(T)} = k \ (k = 0)$	$\texttt{false}$
27	$\texttt{length(T)} > k$	$\texttt{true}$
28	$\texttt{length(T)} < k \ (k \geq 2)$	$\texttt{length(T} - [\texttt{NOW} - l, \texttt{NOW}]) < k$
29	$\texttt{length(T)} < k \ (k = 0, 1)$	$\texttt{false}$

Table 5: $\hat{\alpha}$; $\texttt{T}$ can be a set of periods. (Continued on next page.)

	p	$\hat{\alpha}(p)$
30	T overlaps $[t_1, t_2]$	T overlaps $[t_1, t_2] \vee \mathtt{NOW} - l \leq t_2$
31	T not overlaps $[t_1, t_2]$ $(l > 0)$	T not overlaps $[t_1, t_2] - [\mathtt{NOW} - l, \infty]$
32	T not overlaps $[t_1, t_2]$ $(l = 0)$	T not overlaps $[t_1, t_2] - [\mathtt{NOW}, \infty]$ $\wedge (\mathtt{NOW} > t_2 \vee \mathtt{NOW} < t_1 \vee \mathtt{TS} < t_1)$
33	T overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$	true
34	T not overlaps $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$	true
35	T overlaps $[t, \mathtt{NOW} - k]$	true
36	T not overlaps $[t, \mathtt{NOW} - k]$ $(k > l)$	T not overlaps $[t, \mathtt{NOW} - k]$
37	T not overlaps $[t, \mathtt{NOW} - k]$ $(0 < k \leq l)$	T not overlaps $[t, \mathtt{NOW} - l - 1]$
38	T not overlaps $[t, \mathtt{NOW} - k]$ $(k = 0)$	see Row 18
39	T overlaps $[\mathtt{NOW} - k, t]$ $(k > l)$	T overlaps $[\mathtt{NOW} - k, t] \vee \mathtt{NOW} - l \leq t$
40	T overlaps $[\mathtt{NOW} - k, t]$ $(k \leq l)$	$\mathtt{NOW} - k \leq t$
41	T not overlaps $[\mathtt{NOW} - k, t]$	true
42	T contains $[t_1, t_2]$	T contains $[t_1, t_2] \vee$ T contains $[t_1, \mathtt{NOW} - l - 1]$
43	T not contains $[t_1, t_2]$ $(l > 0)$	T not contains $[t_1, t_2] \vee \mathtt{NOW} - l \leq t_2$
44	T not contains $[t_1, t_2]$ $(l = 0)$	T not contains $[t_1, t_2] \vee (\mathtt{NOW} = t_2 \wedge \mathtt{TS} < \mathtt{NOW})$
45	T contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$	true
46	T not contains $[\mathtt{NOW} - k_1, \mathtt{NOW} - k_2]$	true
47	T contains $[t, \mathtt{NOW} - k]$ $(k > l)$	T contains $[t, \mathtt{NOW} - k]$
48	T contains $[t, \mathtt{NOW} - k]$ $(k \leq l)$	T contains $[t, \mathtt{NOW} - l - 1]$
49	T not contains $[t, \mathtt{NOW} - k]$	true
50	T contains $[\mathtt{NOW} - k, t]$	true
51	T not contains $[\mathtt{NOW} - k, t]$ $(k > l > 0)$	T not contains $[\mathtt{NOW} - k, t] \vee \mathtt{NOW} - l \leq t$
52	T not contains $[\mathtt{NOW} - k, t]$ $(k \leq l, l > 0)$	$\mathtt{NOW} - k \leq t$
53	T not contains $[\mathtt{NOW} - k, t]$ $(l = 0)$	T not contains $[\mathtt{NOW} - k, t] \vee (\mathtt{NOW} = t \wedge \mathtt{TS} < \mathtt{NOW})$

Table 5: $\hat{\alpha}$; T can be a set of periods. (Continued from previous page.)

	p	$\hat{\alpha}(p)$
1	p is nontemporal	same as p
2	$\mathtt{NOW} = t$	$\mathtt{NOW} \leq t$
3	$\mathtt{NOW} > t$	$\mathtt{true}$
4	$\mathtt{NOW} < t$	$\mathtt{NOW} < t$
5	$\mathtt{TS} = t \ (l > 0)$	$\mathtt{TS} = t \vee (\mathtt{NOW} - l \leq t \wedge \mathtt{TS} \geq \mathtt{NOW} - l)$
6	$\mathtt{TS} = t \ (l = 0)$	$\mathtt{TS} = t$
7	$\mathtt{TS} > t \ (l > 0)$	$\mathtt{TS} > t \vee (\mathtt{NOW} - l \leq t \wedge \mathtt{TS} \geq \mathtt{NOW} - l)$
8	$\mathtt{TS} > t \ (l = 0)$	$\mathtt{TS} > t$
9	$\mathtt{TS} < t$	$\mathtt{TS} < t \vee \mathtt{NOW} - l < t$
10	$\mathtt{TS} = \mathtt{NOW} - k \ (k > l)$	$\mathtt{TS} \geq \mathtt{NOW} - k$
11	$\mathtt{TS} = \mathtt{NOW} - k \ (k \leq l)$	$\mathtt{TS} \geq \mathtt{NOW} - l$
12	$\mathtt{TS} > \mathtt{NOW} - k \ (k > l)$	$\mathtt{TS} > \mathtt{NOW} - k$
13	$\mathtt{TS} > \mathtt{NOW} - k \ (0 < k \leq l)$	$\mathtt{TS} \geq \mathtt{NOW} - l$
14	$\mathtt{TS} > \mathtt{NOW} - k \ (k = 0)$	$\mathtt{false}$
15	$\mathtt{TS} < \mathtt{NOW} - k$	$\mathtt{true}$
16	$\mathtt{TE} = t$	$\mathtt{TE} = t$ $\vee (\mathtt{NOW} - l - 1 = t \wedge \mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \leq \mathtt{NOW} - l - 1)$ $\vee (\mathtt{NOW} - l \leq t \wedge \mathtt{TE} \geq \mathtt{NOW} - l - 1)$
17	$\mathtt{TE} > t$	$\mathtt{TE} > t \vee \mathtt{TE} \geq \mathtt{NOW} - l - 1$
18	$\mathtt{TE} < t$	$\mathtt{TE} < t \vee \mathtt{NOW} - l < t \vee (\mathtt{NOW} - 1 = t \wedge \mathtt{TS} < t)$
19	$\mathtt{TE\ is\ NOW}$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
20	$\mathtt{TE\ is\ not\ NOW}$	$\mathtt{true}$
21	$\mathtt{TE} = \mathtt{NOW} - k \ (k > l + 1)$	$\mathtt{TE} \geq \mathtt{NOW} - k$
22	$\mathtt{TE} = \mathtt{NOW} - k \ (k \leq l + 1)$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
23	$\mathtt{TE} > \mathtt{NOW} - k \ (k > l + 1)$	$\mathtt{TE} > \mathtt{NOW} - k$
24	$\mathtt{TE} > \mathtt{NOW} - k \ (0 < k \leq l + 1)$	$\mathtt{TE} \geq \mathtt{NOW} - l - 1$
25	$\mathtt{TE} > \mathtt{NOW} - k \ (k = 0)$	$\mathtt{false}$
26	$\mathtt{TE} < \mathtt{NOW} - k$	$\mathtt{true}$
27	$\mathtt{length(T)} = k \ (k \geq 1)$	$\mathtt{length(T)} = k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} \geq \mathtt{NOW} - l - k)$
28	$\mathtt{length(T)} = k \ (k = 0)$	$\mathtt{false}$
29	$\mathtt{length(T)} > k$	$\mathtt{length(T)} > k \vee \mathtt{TE} \geq \mathtt{NOW} - l - 1$
30	$\mathtt{length(T)} < k \ (k \geq 2)$	$\mathtt{length(T)} < k \vee (\mathtt{TE} \geq \mathtt{NOW} - l - 1 \wedge \mathtt{TS} > \mathtt{NOW} - l - k)$
31	$\mathtt{length(T)} < k \ (k = 0, 1)$	$\mathtt{false}$

Table 6: $\hat{\alpha}$; $\mathtt{T}$ contains a single period.