Maintaining Temporal Views Over Non-Temporal Information Sources For Data Warehousing[*][†]

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

Abstract

An important use of data warehousing is to provide temporal views over the history of source data that may itself be non-temporal. While recent work in view maintenance is applicable to data warehousing, only non-temporal views have been considered. In this paper, we introduce a framework for maintaining temporal views over non-temporal information sources in a data warehousing environment. We describe an architecture for the temporal data warehouse that automatically maintains temporal views over non-temporal source relations, and allows users to ask temporal queries using these views. Because of the dimension of time, a materialized temporal view may need to be updated not only when source relations change, but also as time advances. We present incremental techniques to maintain temporal views for both cases, and outline the implementation of our approach in the WHIPS warehousing prototype at Stanford.

1 Introduction

A data warehouse is a repository for efficient querying and analysis of integrated information from a wide variety of sources. The warehouse effectively maintains materialized views over base relations at the sources [LW95, Wid95]. Clients of the warehouse may not only be interested in the most up-to-date information, but also the history of how the source data has evolved. It is therefore important that the warehouse supports temporal queries. On the other hand, underlying information sources often are non-temporal, *i.e.*, only the current state of the data is available. Even if the source does carry historical information, say, transaction logs or dumps of old database states, using this information for querying may be prohibitively expensive. Thus, we are interested in techniques by which the warehouse can materialize relevant temporal views over the history of non-temporal source data. Many applications will benefit from such a temporal data warehouse. For example, a warehouse that stores daily bank account balances can provide audit logs for financial analysts. A company may use a temporal warehouse to keep track of periodic marketing and income figures, personnel transfers, and the history of relevant corporate partnerships.

There has been little research work to date on the temporal view maintenance problem, in the warehousing context or otherwise. Meanwhile, since warehousing applications do often require

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

[†]This paper is the full version of [YW98].

temporal information, most commercial systems store time information as normal attributes in the standard relational model, and query it using SQL. However, as pointed out by the temporal database community, writing "interesting" temporal queries in the absence of an actual temporal data model can be extremely cumbersome and error-prone. Without custom programming, users cannot express temporal queries whose interpretation may change over time, let alone specifying them as views to be maintained automatically. Furthermore, when time is treated as a normal attribute, it is difficult for these systems to exploit the semantics of time to maintain temporal views. Instead, they usually rely on input from the warehouse administrator to "vacuum" historical data no longer required for online analysis.

To address the needs of applications and overcome the limitations of current systems, we introduce a framework for maintaining temporal views over non-temporal information sources in a data warehousing environment. We propose an architecture for the temporal data warehouse that automatically maintains temporal views over non-temporal source relations, and allows users to ask temporal queries using these views. The temporal view definition language we consider is equivalent to a subset of TSQL2 [SJS95] including a class of commonly used aggregates termed *moving-window aggregates* [NA89]. We present an algorithm to incrementally maintain temporal views as time advances and source relations undergo changes.

Figure 1 depicts a simple temporal data warehouse architecture. Each data source contains a number of non-temporal source relations which are monitored and exported by an *extractor* [Wid95]. When a non-temporal source relation is updated, the extractor notifies the warehouse with the content of the update. Conceptually, for each non-temporal source relation R^{nt}, the warehouse maintains a temporal relation R, which encodes the complete history of R^{nt}. R can then be used as a base relation to construct definitions for materialized temporal views. Users formulate queries over the set of temporal views exported by the warehouse. It is important to note that the temporal base relations at the warehouse serve primarily as *conceptual* base relations to define views; they are not necessarily materialized.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 formalizes the temporal data model and view definition language we use. The procedure for incorporating source updates into the warehouse and the incremental maintenance algorithm for non-aggregate temporal views are presented in Section 4. The extension of our algorithm for temporal aggregates is discussed in Section 5. Section 6 briefly describes how our framework is being integrated into the WHIPS data warehousing prototype at Stanford [WGL+96]. Section 7 concludes and discusses further research. Complete proofs for all theorems are provided in Appendix A.

2 Related Work

The view maintenance problem has been studied extensively; see [GM95] for a survey. However, most work to date considers only non-temporal views. Although temporal view maintenance introduces many problems that are not present for non-temporal views, a number of techniques developed for maintaining non-temporal views are still applicable. For example, we adopt the algebraic approach of [QW91, GL95] for propagating source changes (Section 4.2), and we apply

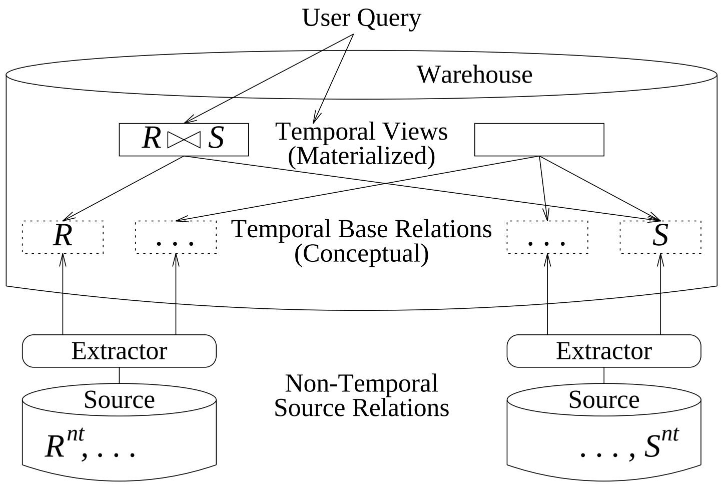


Figure 1: A simple temporal data warehouse architecture.

satisfiability tests to predicates in order to simplify view refresh equations (Section 4.3), similar to the method of detecting *irrelevant updates* in [BCL89]. We also apply methods for maintaining non-temporal aggregate views from [Qua96].

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 [SJS95], as a basis for our work. Our concept of *τ-reducibility* (Section 3.2) is essentially that of *snapshot reducibility* discussed in [Sno87, BJS95]. We use a different name in order to avoid potential confusion with the `SNAPSHOT` construct in TSQL2.

A number of papers are related to temporal view maintenance to some degree. [JMS95] addresses the temporal view maintenance problem in the *chronicle* data model. Under that model, each tuple is timestamped with a single time instant rather than a set; consequently, the algebra is weaker than ours, and more efficient maintenance algorithms are possible. [JMR91] and [JM93] present techniques to incrementally maintain views defined using non-temporal operators over arbitrary snapshots of base relations, rather than temporal views over the complete history. In order to access the complete history, *backlogs* must be queried explicitly. The *differential operators* they introduce are closely related to our change propagation through τ-reducible operators (Section 4.2.1). [BM95] considers the maintenance of relational queries whose selection predicates may refer to the symbol `NOW`, the current time. We consider a more expressive temporal query language and also handle `NOW` stored in data. [CT95] and [Ple93] explore the related problem of monitoring temporal integrity constraints. The languages they use are logical rather than algebraic, and are generally less expressive. Furthermore, the problem of detecting constraint violations is a strict subset of the problem of maintaining materialized views. [BL89] provides some intuition, but no complete algorithms, for temporal view maintenance in a *temporally grouped* data model, while our data model is *temporally ungrouped* [CCT94]. Finally, [LGM97] develops a framework for system-managed expiration of warehouse data, which can be used to vacuum historical data for a restricted class of temporal views.

3 Preliminaries

We present our temporal data model and view definition language in Sections 3.1 and 3.2; both are fairly standard. Section 3.2 provides a classification of the temporal algebra operators which will be useful in developing our algorithms. A running example is introduced in Section 3.3, together with some intuition for the temporal view maintenance problem.

3.1 Temporal Data Model

The temporal data model we use is essentially BCDM (*Bitemporal Conceptual Data Model* [JSS94]) restricted to transaction time. Intuition and examples for the definition will be given below.

Definition 3.1 (Temporal Database) Let $U_D = \{D_1, D_2, ..., D_{n_D}\}$ be a set of non-empty value domains, and let $D = \cup D_i$ be the set of all values. Let $\mathbb{T} = \{t_0, t_1, ...\}$ be a finite, non-empty set of *time instants* (also known as *chronons*) with $<$ as the total order relation, and let $P_{\mathbb{T}} = \{c \mid c \subseteq \mathbb{T} \cup \{\texttt{NOW}\} \wedge c \neq \emptyset\}$. Let $U_A = \{A_1, A_2, ..., A_{n_A}\}$ be a set of *explicit attributes*, and let $\texttt{T}$ be a distinguished *timestamp attribute.*

A *temporal relation schema* is defined as a pair $\langle A, dom \rangle$, where: (1) $A \subseteq U_A$, and $A \cup \{\texttt{T}\}$ is the set of attributes of the schema; (2) A includes a key of the schema, *i.e.*, $A \rightarrow A \cup \{\texttt{T}\}$; (3) *dom* is a function from $A \cup \{\texttt{T}\}$ to $U_D \cup \{P_{\mathbb{T}}\}$ which assigns domains in U_D to attributes in A and $P_{\mathbb{T}}$ to $\texttt{T}$.

A *temporal database schema* is a finite set of temporal relation schemas.

A *tuple* on relation schema $\langle A, dom \rangle$ is a function from $A \cup \{\texttt{T}\}$ to $D \cup P_{\mathbb{T}}$ which assigns an element in $dom(A_i)$ to each attribute $A_i \in A$, and an element in $P_{\mathbb{T}}$ to $\texttt{T}$.

A *temporal relation* with schema $\langle A, dom \rangle$ is a finite set of tuples that satisfy the key constraint $A \rightarrow A \cup \{\texttt{T}\}$.

A *temporal database* at the current (transaction) time t_{now} consists of a set of temporal relations, with the additional constraint that any tuple u in the database satisfies: $\forall t \in u.\texttt{T}$: $t \leq t_{now}$ or t is the symbol $\texttt{NOW}$; furthermore, $t_{now} \in u.\texttt{T}$ if and only if $\texttt{NOW} \in u.\texttt{T}$. □

Intuitively, a temporal relation schema has the form $(A_1, A_2, ..., A_m, \texttt{T})$, where the A_i's are explicit attributes and $\texttt{T}$ is the implicit, set-valued timestamp attribute. A tuple $\langle a_1, a_2, ..., a_m, c \rangle$ is interpreted as follows: The non-temporal tuple $\langle a_1, a_2, ..., a_m \rangle$ is present in the database snapshot at time t if and only if $t \in c$. Each snapshot is purely relational and contains no duplicates because explicit attributes include a key. For ease of formalization and proofs, we have defined the timestamp attribute to be a set of time instants. It might be more intuitive to use a set of time periods (start and end times) instead. However, since our time domain $\mathbb{T}$ is discrete and has a total order, these two definitions are equivalent. In fact, although we use sets of time instants in our formal model to facilitate presentation, we will assume an implementation based on sets of periods, as explained below.

When the special symbol $\texttt{NOW}$ appears in a tuple r's timestamp attribute, it means that r will continue to "stay alive", so conceptually the database will need to update $r.\texttt{T}$ to include new

id	dept	name	office	phone	T
123456	Research	Amy	121	1–2345	[01/01/91, 05/01/92], [01/01/94, 06/01/96)
123456	Research	Amy	151	1–5432	[06/01/96, 12/31/96]
700000	Development	Ben	B07	7–0000	[11/01/96, 05/01/97]
714285	Development	Coy	B17	7–1428	[11/01/96, NOW]

Figure 2: Contents of temporal relation R on 09/30/97.

time instants as time advances. In practice, such timestamp updates need not be done explicitly, because T can be implemented efficiently by storing start and end times for the set of maximal, non-overlapping periods encoding T. One of the end times may store the symbol NOW, which "automatically" extends T as time advances. We will assume this implementation in some of our later discussions. For a detailed discussion of NOW in temporal databases, please refer to [CDI+97].

Figure 2 shows an example temporal relation R(id, dept, name, office, phone, T) that stores employee directory information for a company, with T encoded as a set of periods.

3.2 Temporal View Definition Language

Views in the data warehouse are constructed from the conceptual temporal base relations (recall Figure 1) using temporal algebra operators. To simplify the initial presentation, we introduce the aggregate operator separately later in Section 5. 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 also define a function $h(R, r', A) \stackrel{\text{def}}{=} \{t \mid \exists r \in R : r.A = r'.A \wedge t \in r.\mathtt{T}\}$, which returns the union of the timestamps of all the tuples in R that agree with tuple r' on the attributes in A. The five non-aggregate operators are:

- (**Difference**) $R \Leftrightarrow S \stackrel{\text{def}}{=} \{\langle r.A_R, c\rangle \mid r \in R \wedge c = r.\mathtt{T} \Leftrightarrow h(S, r, A_R) \wedge c \neq \varnothing\}$. Note that difference is only defined for $A_R = A_S$.
- (**Union**) $R \cup S \stackrel{\text{def}}{=} \{\langle u, c\rangle \mid (\exists r \in R : u = r.A_R \wedge c = r.\mathtt{T} \cup h(S, r, A_R)) \vee (\exists s \in S : u = s.A_R \wedge c = s.\mathtt{T} \cup h(R, s, A_R))\}$. Note that union is only defined for $A_R = A_S$.
- (**Projection**) $\pi_{A'}(R) \stackrel{\text{def}}{=} \{\langle r.A', c\rangle \mid r \in R \wedge c = h(R, r, A')\}$. Note that projection is only defined for $A' \subseteq A_R$.
- (**Selection**) $\sigma_p(R) \stackrel{\text{def}}{=} \{r \mid r \in R \wedge p(r)\}$, where p is a quantifier-free selection predicate.
- (**Join**) $\bowtie_p(R_1, ..., R_n) \stackrel{\text{def}}{=} \{\langle r_1.A_{R_1}, ..., r_n.A_{R_n}, c\rangle \mid r_1 \in R_1 \wedge ... \wedge r_n \in R_n \wedge p(r_1, ..., r_n) \wedge c = r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \wedge c \neq \varnothing\}$, where p is a quantifier-free join predicate.

Intersection can be defined as: $R \cap S \stackrel{\text{def}}{=} R - (R - S)$. We assume that $-$ and $\cup$ have the (same) lowest precedence, and associate to the left. Selection and join predicates may reference explicit attributes, constants, and standard comparison and arithmetic operators, as well as referencing the timestamp attribute T, time constants (including the symbol NOW), and "built-in" functions and predicates such as `start(T)` (returns the smallest element in T), `end(T)` (returns the largest element in T), `length(T)`, `contains($1.T, $2.T)`[1], `overlaps($1.T, $2.T)`, *etc.*, with their expected interpretation. The exact predicate language does not affect our temporal view maintenance algorithms, although we do assume that predicates are closed under conjunction and complement.

Our treatment of the symbol `NOW` deserves some further explanation. During query evaluation, all predicates in the query are evaluated with the implicit binding of $\texttt{NOW} = t_{now}$, where t_{now} is the current time (a constant in $\mathbb{T}$). However, when computing the timestamp for a result tuple, the `NOW` stored in input tuples cannot simply be replaced with t_{now}. `NOW` is still treated as a distinct symbol, and thus may appear in the result timestamp. This approach is different from the TSQL2 approach proposed in [BJS95], in which all occurrences of `NOW` in the query and input relations are replaced with t_{now} during query evaluation. Under that approach, `NOW` would never appear in a result relation, which is not a problem in general for query results. However, the information about whether a result tuple has the potential to "stay alive" as time advances is lost, which does pose a problem for the maintenance of materialized views. Thus our slightly different treatment of `NOW` seems to be necessary.

With a temporal data model and algebra, one might be tempted to develop all new machinery for maintaining temporal views. However, we can avoid entirely "reinventing the wheel" by considering how temporal relations and operators can be reduced to their non-temporal counterparts. This approach has two advantages: First, it allows us to reason with temporal relations by reducing the problem to a simpler one that deals only with non-temporal relations, enabling both intuitive reasoning and formal proofs. Second, by exploiting the properties that are preserved in the reduction, it is possible to reuse some techniques developed for non-temporal view maintenance. For temporal operators that do not reference time explicitly, it is easy to find their non-temporal counterparts. On the other hand, for operators that reference time explicitly, the reduction is more complicated and does not preserve all desirable properties. This leads to the classification of temporal operators into *τ-reducible operators* (Section 3.2.1) and *θ-reducible operators* (Section 3.2.2).

3.2.1 τ-Reducible Operators

First, we formalize the notion of a snapshot by defining the operator τ on a temporal relation R as: $\tau_t(R) \overset{\text{def}}{=} \{\langle r.A_R \rangle \mid r \in R \wedge t \in r.\texttt{T}\}$. Intuitively, $\tau_t(R)$ returns a non-temporal relation containing the set of tuples "alive" in R at time t. As a concrete example, $\tau_{\texttt{04/01/97}}(R)$ returns $\{\langle \texttt{700000}, \texttt{Development}, \texttt{Ben}, \texttt{B07}, \texttt{7-0000}\rangle, \langle \texttt{714285}, \texttt{Development}, \texttt{Coy}, \texttt{B17}, \texttt{7-1428}\rangle\}$ for the relation R in Figure 2. Operator τ is not a part of the temporal algebra, but it gives us a natural way to reduce the problem of comparing two temporal relations to that of comparing two non-temporal relations:

Theorem 3.1 Let R and S be relations in a temporal database with current time t_{now}. $R = S$ if and only if $\forall t, t_0 \leq t \leq t_{now}$: $\tau_t(R) =_{nt} \tau_t(S)$, where $=_{nt}$ is standard relational equality.

[1]We use $\$i$ to denote the i-th argument of a join operator.

Proof outline: Follows from the definition for τ and Definition 3.1. (Recall that complete proofs for all theorems appear in Appendix A.) $\Box$

We now define *τ-reducible operators*, and show that they allow us to reuse many equalities known in non-temporal set-based relational algebra. Note the correspondence with *snapshot reducibility* from [Sno87, BJS95].

Definition 3.2 (**τ-Reducible Operator**) Let $R_1, ..., R_n$ be relations in a temporal database with current time t_{now}. A temporal operator *op* is *τ-reducible* with respect to its non-temporal counterpart op^{nt} if and only if $\forall t, t_0 \leq t \leq t_{now}$: $\tau_t(op(R_1, ..., R_n)) =_{nt} op^{nt}(\tau_t(R_1), ..., \tau_t(R_n))$. $\Box$

Theorem 3.2 Let $\mathcal{E}$ and $\mathcal{F}$ be two temporal relational expressions defined using only τ-reducible operators. Let $\mathcal{E}^{nt}$ and $\mathcal{F}^{nt}$ be the non-temporal relational expressions obtained by replacing, in $\mathcal{E}$ and $\mathcal{F}$, each τ-reducible operator by its non-temporal counterpart, and treating each temporal relation symbol as a non-temporal relation symbol. If $\mathcal{E}^{nt} =_{nt} \mathcal{F}^{nt}$ holds in relational algebra, then $\mathcal{E} = \mathcal{F}$ holds in temporal relational algebra.

Proof outline: Follows from Theorem 3.1 and Definition 3.2. We prove that $\tau_t(\mathcal{E}) =_{nt} \tau_t(\mathcal{F})$ for any t by induction on the structure of expressions. τ-reducible operators allow τ to be "pushed in" during the inductive step. $\Box$

Operators that contain no explicit references to time are τ-reducible:

Theorem 3.3 Operator $-$ is τ-reducible with respect to the non-temporal difference operator $-^{nt}$; $\cup$ is τ-reducible with respect to the non-temporal union operator $\cup^{nt}$; $\pi_{A'}$ is τ-reducible with respect to the non-temporal projection operator $\pi^{nt}_{A'}$; σ_p, where p contains no references to `NOW` or `T`, is τ-reducible to the non-temporal selection operator σ^{nt}_p; and $\bowtie_p$, where p contains no references to `NOW` or `T`, is τ-reducible to the non-temporal join operator $\bowtie^{nt}_p$.

Proof outline: Follows from the definitions of these operators. $\Box$

3.2.2 θ-Reducible Operators

Temporal selection and join operators may make explicit references to time in their predicates. Consequently, finding non-temporal counterparts requires a different mapping. Intuitively, we want to be able to treat `T` as a normal attribute in a non-temporal relation. For this purpose, we define the operator θ as follows: $\theta(R) \overset{\text{def}}{=} \{\langle r.A_R, r.\texttt{T}\rangle \mid r \in R\}$. Here, R is a temporal relation, and $\theta(R)$ returns a non-temporal relation whose schema is the same as R, except that attribute `T` is converted to an explicit value attribute, encoded to adhere to first normal form. As a concrete example for this encoding, we may choose to represent `T` by a bit vector of length $|\mathbb{T}|$, where $\mathbb{T}$ is the time domain. A time instant $t_i \in \texttt{T}$ if and only if the i-th bit of the vector is 1; $\texttt{NOW} \in \texttt{T}$ if and only if $t_{now} \in \texttt{T}$ (recall Definition 3.1). Of course, since $|\mathbb{T}|$ is huge, this encoding scheme serves no practical purpose. We only need to demonstrate its existence in order to develop our theory; θ is never used in any maintenance expressions that we present later.

A temporal predicate p over R can be converted into a non-temporal predicate over $\theta(R)$ by replacing, in p, all timestamp constants and all built-in functions and predicates on timestamps with non-temporal counterparts based on the encoding used for `T`. We will abuse notation by omitting this bijective conversion function. For example, in our definition of θ, $r.\mathtt{T}$ is not only used for the set-valued timestamp attribute of R, but also for the corresponding atomic-valued attribute of $\theta(R)$. Similarly, we will use the same predicate symbol to denote both a temporal predicate and its corresponding encoded non-temporal predicate. The intended interpretation in all cases is clear from the context.

Operator θ provides an alternative to τ and a more general way to reduce the problem of comparing two temporal relations to that of comparing two non-temporal relations. However, as we will see later, its properties are not as nice as τ.

Theorem 3.4 Let R and S be relations in a temporal database. $R = S$ if and only if $\theta(R) =_{nt} \theta(S)$.

Proof outline: Follows from the definition for θ and Definition 3.1. □

We now show how temporal selection and join find their non-temporal counterparts through θ.

Theorem 3.5 Let R be a temporal relation and p a selection predicate on R that references `NOW` and/or `T`. Then, σ_p is *θ-reducible* with respect to the non-temporal selection operator σ_p^{nt} in the sense that: $\theta(\sigma_p(R)) =_{nt} \sigma_p^{nt}(\theta(R))$.

Proof outline: Follows from the definitions for θ and σ_p. □

In the next theorem note that temporal join is reduced to the composition of two non-temporal operators (projection and join) rather than a single one. Unlike temporal join, non-temporal join does not automatically intersect the timestamps of joining tuples, so an additional projection is required to achieve the same effect.

Theorem 3.6 Let $R_1, ..., R_n$ be temporal relations and p a join predicate on these relations that references `NOW` and/or `T`. Let $\bowtie_{p'}^{nt}$ be a non-temporal join operator, where p' denotes the predicate $(p \wedge \$1.\mathtt{T} \cap ... \cap \$n.\mathtt{T} \neq \emptyset)$. Let $\pi_{A'}^{nt}$ be a non-temporal projection operator, where $A' = \{\$1.A, ..., \$n.A, \$1.\mathtt{T} \cap ... \cap \$n.\mathtt{T}\}$. Then, $\bowtie_p$ is *θ-reducible* with respect to $\pi_{A'}^{nt} \circ \bowtie_{p'}^{nt}$ in the sense that: $\theta(\bowtie_p(R_1, ..., R_n)) =_{nt} \pi_{A'}^{nt}(\bowtie_{p'}^{nt}(\theta(R_1), ..., \theta(R_n)))$.

Proof outline: Follows from the definitions for θ and $\bowtie_p$. □

Like τ-reducibility, θ-reducibility also allows the reuse of known equalities in non-temporal relational algebra (but with some restrictions); this will become clear when we discuss change propagation for these operators in Section 4.2.2.

3.3 Example

Consider a non-temporal relation R^{nt}(`id`, `dept`, `name`, `office`, `phone`) that stores employee directory information for a company. Another non-temporal relation S^{nt}(`id`, `project`, `supervisor`)

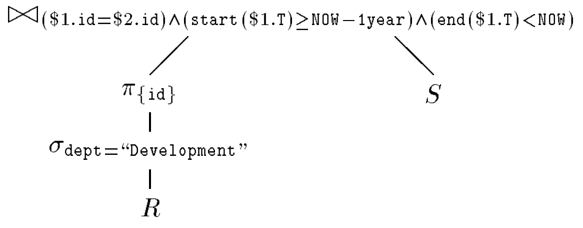


Figure 3: Definition for the temporal view V.

id	dept	name	office	phone	T
123456	Research	Amy	121	1–2345	[01/01/91, 05/01/92], [01/01/94, 06/01/96)
123456	Research	Amy	151	1–5432	[06/01/96, 12/31/96]
700000	Development	Ben	B07	7–0000	[11/01/96, 05/01/97], [10/01/97, NOW]
714285	Development	Coy	B17	7–1428	[11/01/96, 10/01/97)

Figure 4: Contents of temporal relation R on and after 10/01/97.

records current project assignments for all employees. At the temporal data warehouse, there will be two conceptual temporal base relations, R(`id`, `dept`, `name`, `office`, `phone`, T) and S(`id`, `project`, `supervisor`, T), which encode the history of R^{nt} and S^{nt}, respectively. Our previous Figure 2 depicts the state of R on 09/30/97.

Suppose we are interested in a warehouse view that provides the project assignment history of all employees who joined the `Development` department within the past year but are no longer employed within that department now. Using our view definition language, we define:

$$V \overset{\text{def}}{=} \bowtie_{(\$1.\texttt{id}=\$2.\texttt{id})\wedge(\texttt{start}(\$1.\texttt{T})\geq\texttt{NOW}-\texttt{1year})\wedge(\texttt{end}(\$1.\texttt{T})<\texttt{NOW})} \Big(\pi_{\{\texttt{id}\}}(\sigma_{\texttt{dept}=\text{``}\texttt{Development}\text{''}}(R)),\quad S\Big).$$

A parse tree representation of V is shown in Figure 3. Here the projection operator is necessary because there may be multiple tuples in R with the same values for `id` but different values for `office` or `phone`, if some employee changes office or phone number (*e.g.*, `Amy` in Figure 2).

On 09/30/97, only the `Ben` tuple satisfies the join predicate, so V only contains the project assignment history of employee 700000. Suppose that on 10/01/97, `Ben` returns to the company and is assigned the same employee ID, office, and phone: ⟨700000, `Development`, `Ben`, B07, 7–0000⟩ is inserted into R^{nt} and a tuple recording `Ben`'s new project assignment is inserted into S^{nt}. Also, suppose `Coy` leaves the company on the same day: ⟨714285, `Development`, `Coy`, B17, 7–1428⟩ is deleted from R^{nt} and the corresponding tuples are deleted from S^{nt}. A general method for maintaining conceptual temporal base relations will be given in Section 4.1; for now we shall rely on intuition to conclude that after the update, R should contain the tuples shown in Figure 4. For brevity, we do not specify the complete contents of S, since they are not needed at this point to fully understand the example.

Let us consider how these source updates affect the view. `Ben`'s return changes the timestamp of the temporal `Ben` tuple from $\{[\texttt{11/01/96}, \texttt{05/01/97}]\}$ to $\{[\texttt{11/01/96}, \texttt{05/01/97}], [\texttt{10/01/97}, \texttt{NOW}]\}$, which no longer satisfies the predicate $\texttt{end(\$1.T)} < \texttt{NOW}$. As a result, the project assignment history of employee `700000` is removed from V. On the other hand, `Coy`'s departure makes the `Coy` tuple's `T` now satisfy the join predicate, so the project assignment history of employee `714285` is added to V.

An important observation here is that θ-reducible temporal operators, such as the temporal join in our example, do not preserve the monotonicity of their non-temporal counterparts. That is, we cannot guarantee $op(..., R, ...) \subseteq op(..., R \cup R', ...)$. Monotonicity, which is exhibited in all non-temporal operators except the difference operator, is a nice property for view maintenance, because it allows us to conclude that insertions into base relations can only cause insertions, but not deletions, to the view. Monotonicity fails for θ-reducible temporal operators because under θ-reduction, $R \cup R'$ can act as a modification of the attribute `T`, rather than as an insertion, as illustrated above by the `Ben` tuple. $R - R'$ also can result in modification, as illustrated by the `Coy` tuple.

The fact that source updates may affect the view is no surprise. However, a temporal view may change even without source updates, due to the dimension of time. Tuple timestamps containing `NOW` are extended as time advances; for example, `length(T)` may increase over time. Moreover, the interpretation for predicates that reference `NOW` also changes over time. For example, suppose that R^{nt} and S^{nt} stay the same after `10/01/97`. If we evaluate V two months later, on `12/01/97`, we notice that V no longer contains the project assignment history for `Coy`, because the predicate $\texttt{start(T)} \geq \texttt{NOW} - \texttt{1year}$ is no longer satisfied when the binding for `NOW` changes from `10/01/97` to `12/01/97`. This automatic advancement of time is a desirable feature of the temporal algebra, since it allows us to query data as a function of time, without custom programming. However, there is no non-temporal analogy for this phenomenon, so we cannot apply known techniques for non-temporal view maintenance in this case.

4 Maintaining Temporal Views

As we have seen in the example of Section 3.3, temporal views need to be maintained on two occasions: when source relations are updated (we call this *change propagation*), and when the temporal database's current time advances (we call this *view refresh*). We adopt an "eager" approach for change propagation: Changes at sources are immediately propagated through all the affected views. On the other hand, we adopt a "lazy" approach for view refresh: We refresh a materialized view with respect to advanced time only when the view is required for computation. A view is required for computation if it is affected by a source update, or if it is referenced in a user query. Note that a lazy approach for refreshing views is essential to performance. It is clearly unacceptable if the warehouse has to refresh all its materialized views every time the clock ticks.

The complete algorithm for temporal view maintenance is outlined as follows:

1. When some extractor (recall Figure 1) reports a source update at t_{now} that affects the conceptual temporal base relation R:

- Step 1.1: If necessary, the warehouse converts the source update it receives to an update on R in the form of *deltas* $\bigtriangledown R$ and $\triangle R$, in the same style as [QW91, GL95]. This procedure is discussed in Section 4.1.
- Step 1.2: The warehouse refreshes all the views whose definitions reference R so that they are current with respect to t_{now}. This procedure is discussed in Section 4.3.
- Step 1.3: $\bigtriangledown R$ and $\triangle R$ are propagated through all the affected views in the warehouse. This procedure is discussed in Section 4.2.

2. When the user presents a query to the warehouse at time t_{now}:

   - Step 2.1: The warehouse refreshes all the views referenced in the user query so that they are current with respect to t_{now}. This procedure is the same as Step 1.2 above and is discussed in Section 4.3.
   - Step 2.2: Using the refreshed views, the warehouse computes the query results.

4.1 Maintaining Conceptual Temporal Base Relations

For each non-temporal source relation R^{nt}, the warehouse conceptually maintains a temporal relation R which encodes the history of R^{nt}. Ideally, to record precise history, we would like each source update to be timestamped with the time when the update occurs at the source. However, such information often is unavailable, and we may incur a delay from the actual update time to the time when the update is detected by the extractor, and a delay from detection time to the time when the update is reported to the warehouse. Thus, in practice, an update is timestamped at one of the three locations: (1) It may be timestamped at the warehouse with the current transaction time when it arrives; this is necessary if the update as reported by the extractor contains no timestamp information. (2) It may be timestamped at the extractor with the time at which it is detected; then the extractor reports both the content and the timestamp of the update to the warehouse. (3) The timestamp may be provided by the source, *e.g.*, if updates are already logged at the source with timestamps, or if the source actually supports temporal relations (in which case updates are arbitrary and possibly retroactive). In the third scenario, Step 1.1 in our algorithm can be omitted; the rest of the algorithm remains unaffected. Otherwise, the only assumption we make is that the timestamps of updates should be no *later* than the time when they arrive at the warehouse.

Suppose the extractor reports that R^{nt} undergoes the update $R^{nt} \leftarrow R^{nt} -^{nt} \bigtriangledown R^{nt} \cup^{nt} \triangle R^{nt}$, and this update is timestamped by t_{now}. We wish to find $\bigtriangledown R$ and $\triangle R$ such that $R \leftarrow R - \bigtriangledown R \cup \triangle R$ will continue to "faithfully encode" the history of R^{nt}. First we formalize the notion of a faithful history encoding as follows:

Definition 4.1 (**Faithful History Encoding**) Let R be a relation in a temporal database with current time t_{now}. R is a *faithful history encoding* of non-temporal relation R^{nt} if and only if $\forall t, t_0 \leq t \leq t_{now} : \tau_t(R)$ is identical to the state of R^{nt} at t. □

By the definitions of temporal $-$ and $\cup$, it turns out that by simply timestamping $\bigtriangledown R^{nt}$ and $\triangle R^{nt}$ with $\{t_{now}, \texttt{NOW}\}$ we obtain $\bigtriangledown R$ and $\triangle R$ with the desired property:

Theorem 4.1 Suppose that R^{nt} is a non-temporal relation which undergoes the update $R^{nt} \leftarrow R^{nt} -^{nt} \bigtriangledown R^{nt} \cup^{nt} \bigtriangleup R^{nt}$ at time t_{now}. Let R_{old} be a faithful history encoding of R^{nt} immediately before the update. Let $\bigtriangledown R = \{\langle r^{nt}, \{t_{now}, \texttt{NOW}\}\rangle \mid r^{nt} \in \bigtriangledown R^{nt}\}$ and $\bigtriangleup R = \{\langle r^{nt}, \{t_{now}, \texttt{NOW}\}\rangle \mid r^{nt} \in \bigtriangleup R^{nt}\}$. Then $R_{new} = R_{old} - \bigtriangledown R \cup \bigtriangleup R$ is a faithful history encoding of R^{nt} after the update.

Proof outline: Follows from Definition 4.1 and the definitions of temporal $-$ and $\cup$. $\square$

Intuitively, the effect of subtracting $\bigtriangledown R$ with timestamp $\{t_{now}, \texttt{NOW}\}$ is to terminate the NOW-ending timestamp right before t_{now} for each tuple in $\bigtriangledown R^{nt}$. The effect of adding $\bigtriangleup R$ with timestamp $\{t_{now}, \texttt{NOW}\}$ is to create a NOW-ending timestamp for each tuple in $\bigtriangleup R^{nt}$, which will extend automatically from t_{now}. Note again that we do not require R to be materialized at this point. The sole purpose of this procedure is to generate $\bigtriangledown R$ and $\bigtriangleup R$, which are then used to initiate the change propagation procedure for views discussed in Section 4.2.

4.2 Propagating Changes

Given an update to the temporal base relation R in the form of $\bigtriangledown R$ and $\bigtriangleup R$, we need to propagate the effects of the update through all the views whose definitions reference R. This is achieved by repeatedly applying a set of *change propagation equations* in order to factor out the changes in the base relation from the temporal relational expressions defining the views [QW91, GL95]. Change propagation equations for a temporal operator *op* have the following two forms:

$$\begin{aligned} op(R_1, ..., R_i \Leftrightarrow \bigtriangledown R_i, ..., R_n) &= op(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bigtriangledown^- \mathcal{E} \cup \bigtriangleup^- \mathcal{E} \\ op(R_1, ..., R_i \cup \bigtriangleup R_i, ..., R_n) &= op(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bigtriangledown^+ \mathcal{E} \cup \bigtriangleup^+ \mathcal{E} \end{aligned}$$

where $\bigtriangledown^- \mathcal{E}$ and $\bigtriangleup^- \mathcal{E}$ (respectively, $\bigtriangledown^+ \mathcal{E}$ and $\bigtriangleup^+ \mathcal{E}$) are expressions possibly containing $R_1, ..., R_n$ and $\bigtriangledown R_i$ (respectively, $\bigtriangleup R_i$), or may be empty. We present the change propagation equations for the τ-reducible operators in Section 4.2.1 and for the θ-reducible operators in Section 4.2.2.

The change propagation procedure works as follows: Let $\mathcal{E}$ be a temporal relational expression, and let $\delta(\mathcal{E})$ denote the result of propagating $\bigtriangledown R$ and $\bigtriangleup R$ through $\mathcal{E}$. The goal is to derive expressions $\bigtriangledown \mathcal{E}$ and $\bigtriangleup \mathcal{E}$ such that $\delta(\mathcal{E}) = \mathcal{E} - \bigtriangledown \mathcal{E} \cup \bigtriangleup \mathcal{E}$. If $\mathcal{E}$ is the definition for a materialized view, then we can use $\bigtriangledown \mathcal{E}$ and $\bigtriangleup \mathcal{E}$ to maintain the view incrementally. We derive $\bigtriangledown \mathcal{E}$ and $\bigtriangleup \mathcal{E}$ by induction on the structure of $\mathcal{E}$. The base case is simple: $\delta(R) = R - \bigtriangledown R \cup \bigtriangleup R$, and $\delta(S) = S - \varnothing \cup \varnothing$ where S is a base relation other than R. Now follows the inductive step: Suppose that $\mathcal{E} = op(\mathcal{E}_1, ..., \mathcal{E}_n)$. The inductive hypothesis is $\forall i : \delta(\mathcal{E}_i) = \mathcal{E}_i - \bigtriangledown \mathcal{E}_i \cup \bigtriangleup \mathcal{E}_i$. Then,

$$\delta(\mathcal{E}) = op\Big(\delta(\mathcal{E}_1), ..., \delta(\mathcal{E}_n)\Big) = op\Big(\mathcal{E}_1 \Leftrightarrow \bigtriangledown \mathcal{E}_1 \cup \bigtriangleup \mathcal{E}_1, ..., \mathcal{E}_n \Leftrightarrow \bigtriangledown \mathcal{E}_n \cup \bigtriangleup \mathcal{E}_n\Big).$$

By repeatedly applying the change propagation equations for *op*, we will be able to factor out all $\bigtriangledown \mathcal{E}_i$'s and $\bigtriangleup \mathcal{E}_i$'s and combine them into $\bigtriangledown \mathcal{E}$ and $\bigtriangleup \mathcal{E}$ representing the net effect. Thus we arrive at:

$$\delta(\mathcal{E}) = op(\mathcal{E}_1, ..., \mathcal{E}_n) \Leftrightarrow \bigtriangledown \mathcal{E} \cup \bigtriangleup \mathcal{E} = \mathcal{E} \Leftrightarrow \bigtriangledown \mathcal{E} \cup \bigtriangleup \mathcal{E}.$$

4.2.1 τ-Reducible Operators

Based on Theorem 3.2, change propagation equations for τ-reducible operators can be derived directly from the corresponding equalities in non-temporal relational algebra, as in, *e.g.*, [QW91]. The equations are identical in form to their non-temporal counterparts, except that all operators are now temporal.

Theorem 4.2 In the following, p contains no references to `NOW` or `T`.

$$
\begin{array}{rl}
(c.1) & (R \Leftrightarrow \bigtriangledown R) \Leftrightarrow S = (R \Leftrightarrow S) \Leftrightarrow (\bigtriangledown R \Leftrightarrow S) \\
(c.2) & (R \cup \bigtriangleup R) \Leftrightarrow S = (R \Leftrightarrow S) \cup (\bigtriangleup R \Leftrightarrow S) \\
(c.3) & R \Leftrightarrow (S \Leftrightarrow \bigtriangledown S) = (R \Leftrightarrow S) \cup (R \cap \bigtriangledown S) \\
(c.4) & R \Leftrightarrow (S \cup \bigtriangleup S) = (R \Leftrightarrow S) \Leftrightarrow (R \cap \bigtriangleup S) \\
(c.5) & (R \Leftrightarrow \bigtriangledown R) \cup S = (R \cup S) \Leftrightarrow (\bigtriangledown R \Leftrightarrow S) \\
(c.6) & (R \cup \bigtriangleup R) \cup S = (R \cup S) \cup (\bigtriangleup R \Leftrightarrow S) \\
(c.7) & \pi_{A'}(R \Leftrightarrow \bigtriangledown R) = \pi_{A'}(R) \Leftrightarrow (\pi_{A'}(R) \Leftrightarrow \pi_{A'}(R \Leftrightarrow \bigtriangledown R)) \\
(c.8) & \pi_{A'}(R \cup \bigtriangleup R) = \pi_{A'}(R) \cup (\pi_{A'}(\bigtriangleup R) \Leftrightarrow \pi_{A'}(R)) \\
(c.9) & \sigma_p(R \Leftrightarrow \bigtriangledown R) = \sigma_p(R) \Leftrightarrow \sigma_p(\bigtriangledown R) \\
(c.10) & \sigma_p(R \cup \bigtriangleup R) = \sigma_p(R) \cup \sigma_p(\bigtriangleup R) \\
(c.11) & \bowtie_p(R_1, ..., R_i \Leftrightarrow \bigtriangledown R_i, ..., R_n) = \bowtie_p(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bowtie_p(R_1, ..., \bigtriangledown R_i, ..., R_n) \\
(c.12) & \bowtie_p(R_1, ..., R_i \cup \bigtriangleup R_i, ..., R_n) = \bowtie_p(R_1, ..., R_i, ..., R_n) \cup \bowtie_p(R_1, ..., \bigtriangleup R_i, ..., R_n)
\end{array}
$$

Proof: Follows from the τ-reducibility of these operators (Theorems 3.2 and 3.3) and the correctness of the equalities for non-temporal relational algebra. □

4.2.2 θ-Reducible Operators

As motivated in Section 3.3, applying $\bigtriangledown R$ and $\bigtriangleup R$ to R sometimes behaves like (timestamp) modifications rather than deletion and insertions under θ reduction. Therefore, given arbitrary $\bigtriangledown R$ and $\bigtriangleup R$, we cannot simply reuse the non-temporal change propagation equations for θ-reducible operators. Instead, we provide a way in the following theorem to transform $\bigtriangledown R$ and $\bigtriangleup R$ into *pure deltas*, *i.e.*, deltas that do not behave like modifications under θ reduction. Then the known non-temporal equalities can be reused. First, we define the temporal semijoin operator $\ltimes_p$ as: $\ltimes_p(R_1, R_2, ..., R_n) \overset{\text{def}}{=} \{r_1 \mid \exists r_2, ..., r_n : r_1 \in R_1 \wedge r_2 \in R_2 \wedge ... \wedge r_n \in R_n \wedge p(r_1, r_2, ..., r_n)\}$.

Theorem 4.3 Suppose that $R_{new} = R - \bigtriangledown_1 R \cup \bigtriangleup_1 R$. Let $\bigtriangledown_2 R = \ltimes_{\$1.A_R = \$2.A_R}(R, \bigtriangledown_1 R \cup \bigtriangleup_1 R)$ and $\bigtriangleup_2 R = \bigtriangledown_2 R - \bigtriangledown_1 R \cup \bigtriangleup_1 R$. Then: (i) $R_{new} = R - \bigtriangledown_2 R \cup \bigtriangleup_2 R$; (ii) $\theta(R - \bigtriangledown_2 R) =_{nt} \theta(R) -^{nt} \theta(\bigtriangledown_2 R)$; (iii) $\theta(R - \bigtriangledown_2 R \cup \bigtriangleup_2 R) =_{nt} \theta(R - \bigtriangledown_2 R) \cup^{nt} \theta(\bigtriangleup_2 R)$.

Proof outline: Follows from the definitions for θ, $-$, and $\cup$. □

Intuitively, we use semijoin in the definition for $\bigtriangledown_2 R$ to obtain all the complete tuples in R that are affected by $\bigtriangledown_1 R$ and $\bigtriangleup_1 R$. $\bigtriangleup_2 R$ is the result of applying $\bigtriangledown_1 R$ and $\bigtriangleup_1 R$ to these tuples. The notion of pure deltas is captured by Conditions (ii) and (iii) in Theorem 4.3. Condition (ii) ensures that $\bigtriangledown_2 R$ will remove tuples completely, instead of just shrinking their timestamps. Condition (iii) guarantees that $\triangle_2 R$ only contains tuples that do not agree with any existing tuple on the set of explicit attributes, so that it cannot possibly expand any existing tuple's timestamp.

Change propagation equations for θ-reducible operators are presented below in Theorem 4.4. Again, the equations are identical to their non-temporal counterparts, but with an important restriction that $\bigtriangledown R$ and $\triangle R$ must be pure deltas. Before applying the following change propagation equations, we may need to apply Theorem 4.3 to $\bigtriangledown R$ and $\triangle R$ first, if the requirement of pure deltas is not already satisfied.

Theorem 4.4 In the following, p may reference `NOW` and/or `T`. We require that $\theta(R - \bigtriangledown R) =_{nt} \theta(R) -^{nt} \theta(\bigtriangledown R)$ and $\theta(R \cup \triangle R) =_{nt} \theta(R) \cup^{nt} \theta(\triangle R)$ (similarly for $\bigtriangledown R_i$ and $\triangle R_i$).

(c.13) $\sigma_p(R \Leftrightarrow \bigtriangledown R) = \sigma_p(R) \Leftrightarrow \sigma_p(\bigtriangledown R)$
(c.14) $\sigma_p(R \cup \triangle R) = \sigma_p(R) \cup \sigma_p(\triangle R)$
(c.15) $\bowtie_p(R_1, ..., R_i \Leftrightarrow \bigtriangledown R_i, ..., R_n) = \bowtie_p(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bowtie_p(R_1, ..., \bigtriangledown R_i, ..., R_n)$
(c.16) $\bowtie_p(R_1, ..., R_i \cup \triangle R_i, ..., R_n) = \bowtie_p(R_1, ..., R_i, ..., R_n) \cup \bowtie_p(R_1, ..., \triangle R_i, ..., R_n)$

Proof outline: Follows from Theorems 3.4, 3.5, 3.6 and the correctness of known equalities for non-temporal relational algebra. For each equality, we prove that the result of applying θ to the left-hand side is the same as applying θ to the right-hand side. The requirement of pure deltas allows θ to be "pushed in", reducing the equality to non-temporal relational algebra. □

It is also possible to develop change propagation equations for θ-reducible operators without requiring pure deltas. However, doing so significantly complicates the expressions. Furthermore, for the change propagation equations above, it can be shown that the deltas generated by the right-hand sides are indeed pure themselves. If they need to be propagated further through enclosing θ-reducible operators, it is more efficient to use these simple change propagation equations where the requirement of pure deltas is already met.

4.2.3 Example

Consider the temporal view defined in Section 3.3:

$$V \overset{\text{def}}{=} \bowtie_{(\$1.\texttt{id}=\$2.\texttt{id})\wedge(\texttt{start}(\$1.\texttt{T})\geq\texttt{NOW}-\texttt{1year})\wedge(\texttt{end}(\$1.\texttt{T})<\texttt{NOW})} \Big(\pi_{\{\texttt{id}\}}(\sigma_{\texttt{dept}=\text{``}\texttt{Development}\text{''}}(R)),\ \ S\Big).$$

For brevity, let p denote the join predicate $(\$1.\texttt{id} = \$2.\texttt{id}) \wedge (\texttt{start}(\$1.\texttt{T}) \geq \texttt{NOW} - \texttt{1year}) \wedge (\texttt{end}(\$1.\texttt{T}) < \texttt{NOW})$, and let q denote the selection predicate $\texttt{dept} = \text{``}\texttt{Development}\text{''}$.

We show how to derive the maintenance expression for V when $\bigtriangledown R^{nt}$ is deleted from R^{nt} (the case for $\triangle R^{nt}$ is similar). By Theorem 4.1, we obtain the temporal delta: $\bigtriangledown R = \{\langle r^{nt}, \{t_{now}, \texttt{NOW}\}\rangle \mid r^{nt} \in \bigtriangledown R^{nt}\}$. Then,

$$\begin{aligned}
& \bowtie_p \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R \Leftrightarrow \bigtriangledown R)),\ \ S\Big) && (1)\\
\overset{(c.9)}{=}\ & \bowtie_p \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R) \Leftrightarrow \sigma_q(\bigtriangledown R)),\ \ S\Big)\\
\overset{(c.7)}{=}\ & \bowtie_p \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)) \Leftrightarrow \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)) \Leftrightarrow \pi_{\{\texttt{id}\}}(\sigma_q(R) \Leftrightarrow \sigma_q(\bigtriangledown R))\Big),\ \ S\Big).
\end{aligned}$$

As a shorthand, let $U = \pi_{\{\mathtt{id}\}}(\sigma_q(R)) - \pi_{\{\mathtt{id}\}}(\sigma_q(R) - \sigma_q(\bigtriangledown R))$. According to Theorem 4.4, since $\bowtie_p$ is a θ-reducible operator, we must first use Theorem 4.3 on U to obtain pure deltas:

$$U_1 = \ltimes_{\$1.\mathtt{id}=\$2.\mathtt{id}}\Big(\pi_{\{\mathtt{id}\}}(\sigma_q(R)), \quad U\Big), \quad \text{and} \quad U_2 = U_1 \Leftrightarrow U.$$

Then we continue applying change propagation equations:

$$\begin{aligned}(1) \quad &= \quad \bowtie_p\Big(\pi_{\{\mathtt{id}\}}(\sigma_q(R)) \Leftrightarrow U_1 \cup U_2, \quad S\Big) \\ &\overset{(\mathrm{c}.16),(\mathrm{c}.15)}{=} \quad \bowtie_p\Big(\pi_{\{\mathtt{id}\}}(\sigma_q(R)), \quad S\Big) \Leftrightarrow \bowtie_p(U_1, S) \cup \bowtie_p(U_2, S).\end{aligned}$$

In the above equation, $\bowtie_p\big(\pi_{\{\mathtt{id}\}}(\sigma_q(R)), \quad S\big)$ is exactly the contents of V before the source update; $\bowtie_p(U_1, S)$ and $\bowtie_p(U_2, S)$ represent the incremental changes to V caused by $\bigtriangledown R^{nt}$.

4.3 Refreshing Views

As shown by the example in Section 3.3, the result of evaluating a temporal relational expression may change as time advances, even if none of the base relations has changed. We have chosen a "lazy" approach for refreshing views with respect to the current time. For each materialized view in the warehouse, we record the time at which it was last refreshed. The view is refreshed again only when it is affected by a source update or referenced in a user query.

Before we describe the procedure for refreshing views with respect to time, we need some additional notation: Let $\mathcal{E}$ be a temporal relational expression, and let $[\mathcal{E}]_t$ denote the result of evaluating $\mathcal{E}$ in the temporal database state at time t. We omit the bracket notation for $t = t_{now}$, *i.e.*, $[\mathcal{E}]_{t_{now}} = \mathcal{E}$. We use $\varphi([\mathcal{E}]_t)$ to denote the result of refreshing $[\mathcal{E}]_t$ with respect to t_{now}; thus, $\varphi([\mathcal{E}]_t) = \mathcal{E}$. We also define a special operator ψ_t as: $\psi_t(R) \overset{\mathrm{def}}{=} \{\langle r.A_R, c\rangle \mid r \in R \wedge ((\mathtt{NOW} \notin r.\mathtt{T} \wedge c = r.\mathtt{T}) \vee (\mathtt{NOW} \in r.\mathtt{T} \wedge c = r.\mathtt{T} \cup c'))\}$, where R is a temporal relation and $c' = \{t' \mid t' \in \mathbb{T} \wedge t < t' \leq t_{now}\}$. Intuitively, ψ_t adds the missing time instants between t and t_{now} to tuple timestamps containing `NOW`. This operator is introduced primarily for the purpose of our discussion. As discussed earlier, in reality the timestamp attribute would be implemented as a set of maximal, non-overlapping time periods, and a period ending with `NOW` requires no update as time advances. Therefore, ψ_t is the identity operator in practice.

The procedure for refreshing views with respect to time works as follows: Suppose $\mathcal{E}$ defines a temporal view which was last refreshed at time t. We derive expressions $\bigtriangledown\mathcal{E}$ and $\bigtriangleup\mathcal{E}$ such that $\varphi([\mathcal{E}]_t) = \psi_t([\mathcal{E}]_t) - \bigtriangledown\mathcal{E} \cup \bigtriangleup\mathcal{E}$. To refresh the view, we simply compute $\bigtriangledown\mathcal{E}$ and $\bigtriangleup\mathcal{E}$ and apply them to the already materialized $[\mathcal{E}]_t$. $\bigtriangledown\mathcal{E}$ and $\bigtriangleup\mathcal{E}$ are derived by induction on the structure of $\mathcal{E}$. The base case is trivial: $\varphi([R]_t) = \psi_t([R]_t) - \varnothing \cup \varnothing$ for any base relation R. Suppose that $\mathcal{E} = op(\mathcal{E}_1, ..., \mathcal{E}_n)$. The inductive hypothesis is $\forall i : \varphi([\mathcal{E}_i]_t) = \psi_t([\mathcal{E}_i]_t) - \bigtriangledown\mathcal{E}_i \cup \bigtriangleup\mathcal{E}_i$. Then,

$$\varphi([\mathcal{E}]_t) = op\Big(\varphi([\mathcal{E}_1]_t), ..., \varphi([\mathcal{E}_n]_t)\Big) = op\Big(\psi_t([\mathcal{E}_1]_t) \Leftrightarrow \bigtriangledown\mathcal{E}_1 \cup \bigtriangleup\mathcal{E}_1, ..., \psi_t([\mathcal{E}_n]_t) \Leftrightarrow \bigtriangledown\mathcal{E}_n \cup \bigtriangleup\mathcal{E}_n\Big).$$

By repeatedly applying the change propagation equations for *op* given in Section 4.2, we can factor out all $\bigtriangledown\mathcal{E}_i$'s and $\bigtriangleup\mathcal{E}_i$'s and combine them into $\bigtriangledown\mathcal{E}'$ and $\bigtriangleup\mathcal{E}'$. Thus we have:

$$\varphi([\mathcal{E}]_t) = op\Big(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)\Big) \Leftrightarrow \bigtriangledown\mathcal{E}' \cup \bigtriangleup\mathcal{E}'.$$

We then apply the *view refresh equations* to be given in Sections 4.3.1 and 4.3.2, and arrive at:

$$\varphi([\mathcal{E}]_t) = \Big(\psi_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \Leftrightarrow \nabla\mathcal{E}'' \cup \Delta\mathcal{E}''\Big) \Leftrightarrow \nabla\mathcal{E}' \cup \Delta\mathcal{E}' = \psi_t([\mathcal{E}]_t) \Leftrightarrow \nabla\mathcal{E} \cup \Delta\mathcal{E},$$

where $\nabla\mathcal{E} = \nabla\mathcal{E}'' \cup (\nabla\mathcal{E}' - \Delta\mathcal{E}'')$ and $\Delta\mathcal{E} = (\Delta\mathcal{E}'' - \nabla\mathcal{E}') \cup \Delta\mathcal{E}'$ are the final net effect.

4.3.1 τ-Reducible Operators

Since τ-reducible operators make no explicit references to the timestamp attribute, we would expect that it should be very easy to refresh views whose definitions contain only these operators. Indeed, as shown by the following theorem, refreshing such views involves nothing more than applying ψ_t to the previously materialized views, *i.e.*, deltas generated by this process are empty.

Theorem 4.5 Let *op* be a τ-reducible operator. Then:

$$op(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)) = \psi_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t).$$

Specifically, assume p contains no references to `NOW` or `T`, then:

$$\begin{array}{ll}
\text{(r.1)} & \psi_t([\mathcal{E}_1]_t) \Leftrightarrow \psi_t([\mathcal{E}_2]_t) = \psi_t([\mathcal{E}_1 \Leftrightarrow \mathcal{E}_2]_t) \\
\text{(r.2)} & \psi_t([\mathcal{E}_1]_t) \cup \psi_t([\mathcal{E}_2]_t) = \psi_t([\mathcal{E}_1 \cup \mathcal{E}_2]_t) \\
\text{(r.3)} & \pi_{A'}(\psi_t([\mathcal{E}]_t)) = \psi_t([\pi_{A'}(\mathcal{E})]_t) \\
\text{(r.4)} & \sigma_p(\psi_t([\mathcal{E}]_t)) = \psi_t([\sigma_p(\mathcal{E})]_t) \\
\text{(r.5)} & \bowtie_p(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)) = \psi_t([\bowtie_p(\mathcal{E}_1, ..., \mathcal{E}_n)]_t)
\end{array}$$

Proof outline: Follows from Theorems 3.1 and 3.3 and Definition 3.2. □

4.3.2 θ-Reducible Operators

The predicate p in σ_p and $\bowtie_p$ can explicitly reference the symbol `NOW` and the timestamp attribute `T`. As we have seen in the example of Section 3.3, the truth value of p may not remain the same as time advances. Therefore, refreshing a view whose definition contains σ_p or $\bowtie_p$ may require updating the previously materialized result, as shown by the view refresh equations in the following theorem:

Theorem 4.6 In the following, p may reference `NOW` and/or `T`. In (r.7), q denotes the predicate $\$1.\mathtt{T} \cap ... \cap \$n.\mathtt{T} \neq \emptyset$. The temporal semijoin operator $\ltimes_p$ was defined in Section 4.2.2.

$$\begin{array}{ll}
\text{(r.6)} & \sigma_p(\psi_t([\mathcal{E}]_t)) = \psi_t([\sigma_p(\mathcal{E})]_t) \Leftrightarrow \sigma_{\neg p}\Big(\psi_t([\sigma_p(\mathcal{E})]_t)\Big) \cup \sigma_p\Big(\psi_t([\sigma_{\neg p}(\mathcal{E})]_t)\Big) \\
\text{(r.7)} & \bowtie_p(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)) = \psi_t([\bowtie_p(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \\
& \qquad \Leftrightarrow \bowtie_{\neg p}\Big(\psi_t([\ltimes_{p \wedge q}(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n)]_t), ..., \psi_t([\ltimes_{p \wedge q}(\mathcal{E}_n, \mathcal{E}_1, ..., \mathcal{E}_{n-1})]_t)\Big) \\
& \qquad \cup \bowtie_p\Big(\psi_t([\ltimes_{\neg p \wedge q}(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n)]_t), ..., \psi_t([\ltimes_{\neg p \wedge q}(\mathcal{E}_n, \mathcal{E}_1, ..., \mathcal{E}_{n-1})]_t)\Big)
\end{array}$$

Proof outline: Follows from Equality (c.14) in Theorem 4.4 and the definitions of σ_p and $\bowtie_p$. □

Intuitively, in order to obtain result tuples that satisfy p at t_{now} given the result materialized at t, we first delete tuples that satisfy p at t but not at t_{now}, and then insert tuples that satisfy p at t_{now} but not at t. (r.7) is particularly complex because to be able to reevaluate the join predicate at t_{now}, we need to retain the individual tuple timestamps using semijoin. Without semijoin, we would have lost track of these individual timestamps, since temporal join stores in the result only the intersection of timestamps from joining tuples.

The updates generated by the above view refresh equations appear quite complicated. However, by making some "compile-time" analysis of the predicates, it is possible to determine whether certain deletions and/or insertions are guaranteed to be empty regardless of the database instance. If an update is guaranteed to be empty, then it does not have to be computed or applied at run-time. As an example, consider the deletion $\sigma_{\neg p}\Big(\psi_t([\sigma_p(\mathcal{E})]_t)\Big)$ generated by (r.6). Since we assume an implementation in which a timestamp is encoded by a set of periods using `NOW`, the operator ψ_t can be eliminated. Let $[p]_t$ denote the predicate p evaluated at time t, *i.e.*, with the implicit binding `NOW` $= t$. Then the deletion can be rewritten as $\sigma_{[\neg p]_{t_{now}} \wedge [p]_t}([\mathcal{E}]_t)$. Similarly, it is easy to verify that the deletion in (r.7) can be rewritten as $\bowtie\!\!\lhd_{[\neg p]_{t_{now}} \wedge [p]_t}([\mathcal{E}_1]_t, ..., [\mathcal{E}_n]_t)$. The problem of determining whether these deletions are guaranteed to be empty is reduced to that of testing whether the predicate $[\neg p]_{t_{now}} \wedge [p]_t \wedge t < t_{now}$ is unsatisfiable. If we impose a further restriction that a timestamp `T` can be encoded by just one start time `TS` and one end time `TE` (such an assumption often holds for a transaction time relation), then any temporal predicate, including any built-in function or predicate such as `start`, `length`, or `overlaps`, can be rewritten as nothing more than an ordinary boolean expression involving `TS`, `TE`, $+, -, =, >, <$. Then, existing algorithms [BCL89] can be used to test for satisfiability. A concrete example of using this test appears near the end of Section 4.3.3.

4.3.3 Example

We now demonstrate how to derive the expression for refreshing a temporal view as time advances. We use the same view $V \overset{\text{def}}{=} \bowtie\!\!\lhd_p\big(\pi_{\{\texttt{id}\}}(\sigma_q(R)),\ \ S\big)$ as in Sections 3.3 and 4.2.3. In the following, the operator ψ_t is omitted because we assume an implementation in which a timestamp is encoded by a set of periods using `NOW`.

$$\begin{aligned}
& \varphi(\Big[\bowtie\!\!\lhd_p\Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)),\ \ S\Big)\Big]_t) \qquad (2)\\
= \quad & \bowtie\!\!\lhd_p\Big(\varphi([\pi_{\{\texttt{id}\}}(\sigma_q(R))]_t),\ \ \varphi([S]_t)\Big)\\
\overset{(\text{r.4}),(\text{r.3})}{=} \quad & \bowtie\!\!\lhd_p\Big([\pi_{\{\texttt{id}\}}(\sigma_q(R))]_t,\ \ [S]_t\Big)\\
\overset{(\text{r.7})}{=} \quad & \Big[\bowtie\!\!\lhd_p\Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)),\ \ S\Big)\Big]_t\\
& \ominus \bowtie\!\!\lhd_{[\neg p]_{t_{now}} \wedge [p]_t}\Big([\pi_{\{\texttt{id}\}}(\sigma_q(R))]_t,\ \ [S]_t\Big)\\
& \cup \bowtie\!\!\lhd_{[p]_{t_{now}} \wedge [\neg p]_t}\Big([\pi_{\{\texttt{id}\}}(\sigma_q(R))]_t,\ \ [S]_t\Big).
\end{aligned}$$

In order to test whether the updates generated by (r.7) are empty, we have rewritten them using the $[\,]_t$ notation for predicates. It can be shown easily that the join predicate for the insertion,

$[p]_{t_{now}} \wedge [\neg p]_t$, is unsatisfiable because $(\texttt{start}(\$1.\texttt{T}) \geq t_{now} - \texttt{1year}) \wedge (\texttt{start}(\$1.\texttt{T}) < t - \texttt{1year}) \wedge t < t_{now}$ is unsatisfiable. Therefore, we can simplify (2) into:

$$\begin{aligned}
(2) \quad &= \quad \Big[\bowtie_p \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)),\ \ S \Big) \Big]_t \Leftrightarrow\!\bowtie_{[\neg p]_{t_{now}} \wedge [p]_t} \Big([\pi_{\{\texttt{id}\}}(\sigma_q(R))]_t,\ \ [S]_t \Big) \\
&= \quad \Big[\bowtie_p \Big(\pi_{\{\texttt{id}\}}(\sigma_q(R)),\ \ S \Big) \Big]_t \\
& \qquad \Leftrightarrow\!\bowtie_{\neg p} \Big([\ltimes_{p \wedge \$1.\texttt{T} \cap \$2.\texttt{T} \neq \varnothing}(\pi_{\{\texttt{id}\}}(\sigma_q(R)), S)]_t,\ \ [\ltimes_{p \wedge \$1.\texttt{T} \cap \$2.\texttt{T} \neq \varnothing}(S, \pi_{\{\texttt{id}\}}(\sigma_q(R)))]_t \Big).
\end{aligned}$$

From this equation, we see that the advancement of time automatically "expires" tuples from V. Furthermore, the deletions can be computed using the information in the old V, plus individual timestamps of the joining tuples; this is exactly the purpose of semijoins in the equation above. If we materialize the semijoin results, or simply record individual timestamps of the joining tuples together with V, then there is no need to use the complete source history in R and S.

5 Maintaining Temporal Aggregates

In this section we address the problem of maintaining a class of commonly used temporal aggregate views called *moving-window aggregates* [NA89]. We use $\Pi_{A',w}$ to denote the aggregate operator, where A' is a set of projected and/or aggregated attributes, and $w \geq 0$ is the *window*. The set of regular (projected) attributes in A', which we denote $GB(A')$, specifies the group-by attributes for the aggregate. To facilitate the formal definition of $\Pi_{A',w}$, we first extend the timeslice operator τ_t with an additional parameter for window length: $\tau_{t,w}(R) \overset{\text{def}}{=} \{\langle r.A_R \rangle \mid r \in R \wedge (\exists t' \in r.\text{T} : t - w \leq t' \leq t)\}$. In other words, $\tau_{t,w}(R)$ returns a non-temporal relation containing all tuples valid at some point during the period $[t - w, t]$. Now, we define $\Pi_{A',w}(R)$ as a temporal relation with the property that for any $t \leq t_{now}$, $\tau_t(\Pi_{A',w}(R)) = \Pi^{nt}_{A'}(\tau_{t,w}(R))$. Here $\Pi^{nt}_{A'}$ is the standard non-temporal grouping and aggregation operator [GHQ95, Qua96]. Intuitively, then, the result of a temporal aggregate is a sequence of values generated by moving a window of given length along the time line, and computing the non-temporal aggregate from all tuples valid in the current window. For detailed discussion on temporal aggregates please refer to [SGM93].

As a concrete example, suppose that we have the following temporal relation:

$$R \quad = \quad \Big\{ \langle a, \{[\mathsf{90}, \mathsf{90}], [\mathsf{95}, \mathsf{96}]\} \rangle, \langle b, \{[\mathsf{94}, \mathsf{95}]\} \rangle \Big\}.$$

Then, at $t_{now} = \mathsf{97}$, we compute the `count` aggregate function over R with window length of 0 and 2, respectively. Let t_0 be some known initial time, such as the creation time of relation R.

$$\begin{aligned}
\Pi_{\{\texttt{count}(*)\},0}(R) \quad &= \quad \Big\{ \langle 0, \{[t_0, \mathsf{89}], [\mathsf{91}, \mathsf{93}], [\mathsf{97}, \texttt{NOW}]\} \rangle, \langle 1, \{[\mathsf{90}, \mathsf{90}], [\mathsf{94}, \mathsf{94}], [\mathsf{96}, \mathsf{96}]\} \rangle, \langle 2, \{[\mathsf{95}, \mathsf{95}]\} \rangle \Big\}; \\
\Pi_{\{\texttt{count}(*)\},2}(R) \quad &= \quad \Big\{ \langle 0, \{[t_0, \mathsf{89}], [\mathsf{93}, \mathsf{93}]\} \rangle, \langle 1, \{[\mathsf{90}, \mathsf{92}], [\mathsf{94}, \mathsf{94}]\} \rangle, \langle 2, \{[\mathsf{95}, \texttt{NOW}]\} \rangle \Big\}.
\end{aligned}$$

The situation is depicted graphically in Figure 5 (ignore the EX_2 portion for now). For example, consider the results of the two aggregates at time 92. $\Pi_{\{\texttt{count}(*)\},0}(R)$ evaluates to 0, since there is no tuple in R valid during $[\mathsf{92} - \mathsf{0}, \mathsf{92}]$, *i.e.*, 92. On the other hand, $\Pi_{\{\texttt{count}(*)\},2}(R)$ evaluates to 1, because tuple a is valid in 90, which still falls within the window $[\mathsf{92} - \mathsf{2}, \mathsf{92}]$.

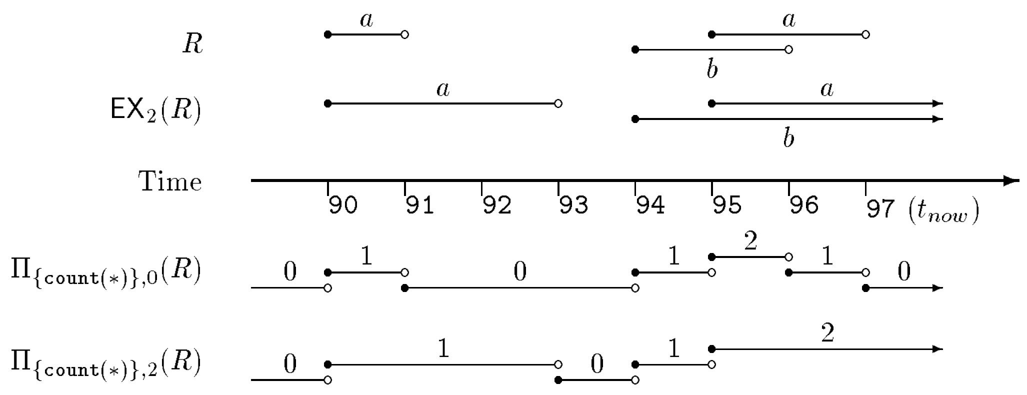


Figure 5: Moving-window aggregates.

5.1 Instantaneous Aggregates

An aggregate with $w = 0$ is termed *instantaneous* because its value at time t is computed only from the tuples valid at time t. By definition, $\Pi_{A',0}$ is τ-reducible: $\tau_t(\Pi_{A',0}(R)) = \Pi^{nt}_{A'}(\tau_{t,0}(R)) = \Pi^{nt}_{A'}(\tau_t(R))$. According to Theorem 3.2, then, for instantaneous aggregates we may simply reuse the change propagation equations developed for non-temporal aggregates [Qua96]:

Theorem 5.1 In the following, we require that $\triangledown R - R = \varnothing$ and $\triangle R \cap R = \varnothing$; that is, we do not attempt to delete nonexistent tuples or insert existing ones. Functions g and f compute new values of aggregate functions based on their old values in $\Pi_{A',0}(R)$ and the results of applying the aggregate functions to the incremental changes $\Pi_{A',0}(\triangledown R)$ and $\Pi_{A',0}(\triangle R)$, respectively. These functions are identical to the ones used in [Qua96] for non-temporal aggregates, and are summarized in Figure 6 for reference. Note that \$2 represents the old aggregate values, and \$1 represents aggregate values calculated from the incremental changes.

(c.17) $\Pi_{A',0}(R \Leftrightarrow \triangledown R) \;=\; \Pi_{A',0}(R) \;\Leftrightarrow\; \pi_{\$2.A}(J) \;\cup\; \pi_{\{g(\$2.a,\$1.a) \mid a \in A'\}}\big(\sigma_{\$2.\mathtt{count}(*)-\$1.\mathtt{count}(*)>0}(J)\big)$,
where $J = \bowtie_{\$1.GB(A')=\$2.GB(A')}\big(\Pi_{A',0}(\triangledown R),\ \Pi_{A',0}(R)\big)$

(c.18) $\Pi_{A',0}(R \cup \triangle R) \;=\; \Pi_{A',0}(R) \;\Leftrightarrow\; \pi_{\$2.A}(J) \;\cup\; \pi_{\{f(\$2.a,\$1.a) \mid a \in A'\}}(J) \;\cup\; \big(\Pi_{A',0}(\triangle R) \Leftrightarrow \pi_{\$1.A}(J)\big)$,
where $J = \bowtie_{\$1.GB(A')=\$2.GB(A')}\big(\Pi_{A',0}(\triangle R),\ \Pi_{A',0}(R)\big)$

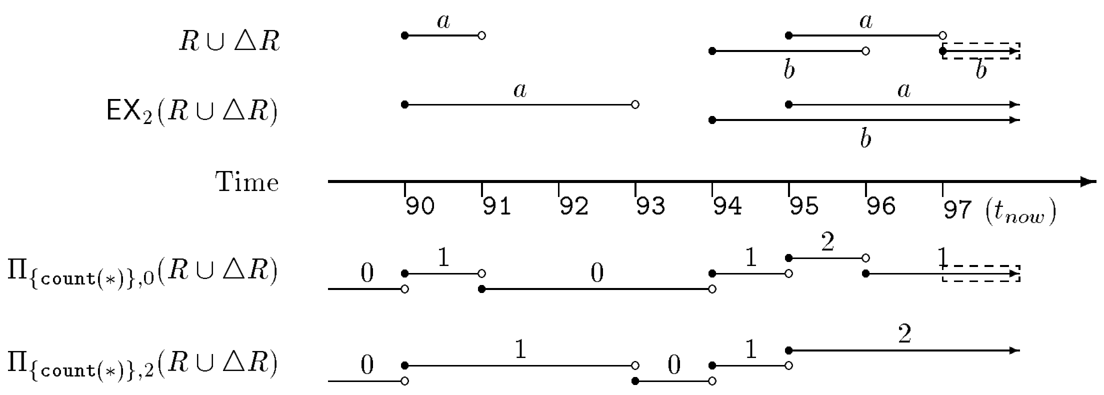
Proof: Follows from the τ-reducibility of $\Pi_{A',0}$, Theorem 3.2, and the correctness of the equalities for non-temporal relational algebra. □

In Theorem 5.1, the requirement that $\triangledown R - R = \varnothing$ is analogous to the notion of *weak minimality* introduced in [GL95]. The requirement that $\triangle R \cap R = \varnothing$ is necessary because our temporal algebra is τ-reduced to *set* algebra. If part of $\triangle R$ intersects with R, then the tuples in the intersection would be counted more than once in the final aggregate. Both requirements can easily be enforced by the change propagation process.

To handle deletions, we assume that `count(*)` is always included in A'. If A' includes `avg`(x), we assume that A' also includes `count(*)` and `sum`(x). Note from Figure 6 that we do not address the problem of maintaining `min` and `max` when deletions occur, since the problem is not unique to

Definition of attribute a	$g(\$2.a, \$1.a)$	$f(\$2.a, \$1.a)$
group-by attribute	$\$2.a$	$\$2.a$
`count`$(*)$	$\$2.a \Leftrightarrow \$1.a$	$\$2.a + \$1.a$
`sum`(x)	$\$2.a \Leftrightarrow \$1.a$	$\$2.a + \$1.a$
`avg`(x)	$\frac{\$2.\texttt{sum}(x) - \$1.\texttt{sum}(x)}{\$2.\texttt{count}(*) - \$1.\texttt{count}(*)}$	$\frac{\$2.\texttt{sum}(x) + \$1.\texttt{sum}(x)}{\$2.\texttt{count}(*) + \$1.\texttt{count}(*)}$
`min`(x)	undefined	`min`$(\$2.a, \$1.a)$
`max`(x)	undefined	`max`$(\$2.a, \$1.a)$

Figure 6: Definition of functions g and f.



$R \cup \triangle R$
$\mathsf{EX}_2(R \cup \triangle R)$
Time
$\Pi_{\{\texttt{count}(*)\},0}(R \cup \triangle R)$
$\Pi_{\{\texttt{count}(*)\},2}(R \cup \triangle R)$
a
b
90
91
92
93
94
95
96
97 (t_{now})
0
1
2

Figure 7: Effect of insertion on moving-window aggregates.

temporal views. In general, more intuition behind the change propagation equations for aggregates can be found in [Qua96]. As a short example, Figure 7 shows the result of propagating an insertion $\triangle R = \{\langle b, \{[97, \texttt{NOW}]\}\rangle\}$ through $\Pi_{\{\texttt{count}(*)\},0}(R)$. (Again, ignore the EX_2 portion for now.) In (c.18), the term $\pi_{\$2.A}(J)$ computes the old aggregate value $\langle 0, \{[97, \texttt{NOW}]\}\rangle$ to be deleted from $\Pi_{\{\texttt{count}(*)\},0}(R)$, and $\pi_{\{f(\$2.a,\$1.a) | a \in A'\}}(J)$ computes the new value $\langle 1, \{[97, \texttt{NOW}]\}\rangle$ to be inserted. Notice that it is unnecessary to access R for the purpose of computing (c.17) and (c.18).

Since $\Pi_{A',0}$ is τ-reducible, its view refresh equation is trivial and follows directly from Theorem 4.5:

$$\text{(r.8)} \quad \Pi_{A',0}(\psi_t([\mathcal{E}]_t)) = \psi_t([\Pi_{A',0}(\mathcal{E})]_t)$$

5.2 Cumulative Aggregates

In the general case where $w > 0$, the aggregate operator $\Pi_{A',w}$ is not τ-reducible. It is termed *cumulative* since its value at time t may depend on tuples that have been valid in the past, as well as those valid at t. To gain some intuition on how change propagation works, consider the effect of the insertion illustrated in Figure 7. For the τ-reducible aggregate $\Pi_{\{\texttt{count}(*)\},0}(R)$, the affected period is exactly the same as the timestamp of $\triangle R$. In contrast, $\Pi_{\{\texttt{count}(*)\},2}(R)$ remains unaffected by $\triangle R$. This is because tuple b in R is valid during $[94, 95]$. Thus, with $w = 2$, it contributes to the value of $\Pi_{\{\texttt{count}(*)\},2}(R)$ during $[94, 95+2]$. Although b is inserted again in 97, its effect is still "shadowed" by the previous valid period of b. Deletions may exhibit similar behavior.

One possible approach is to θ-reduce $\Pi_{A',w}$ and require $\bigtriangledown R$ and $\triangle R$ to be pure deltas, as in Theorem 4.4. This approach avoids the "shadowing" problem, but it causes the aggregate to be recomputed for *all* valid periods of tuple b. This recomputation is both inefficient and unnecessary, because during those periods that are more than w away from the timestamps of $\bigtriangledown R$ and $\triangle R$, the aggregate result cannot be affected. Therefore, we take a different approach based on the following key observation:

$$\Pi_{A',w}(R) = \Pi_{A',0}(\mathsf{EX}_w(R)).$$

Here, EX_w is defined as: $\mathsf{EX}_w(R) \stackrel{\text{def}}{=} \{\langle r.A_R, c\rangle \mid r \in R\}$, where $c = \{t \mid (t \in \mathbb{T} \wedge t \leq t_{now} \wedge (\exists t' \in r.\texttt{T} : 0 \leq t - t' \leq w)) \vee (t = \texttt{NOW} \wedge (\exists t' \in r.\texttt{T} : 0 \leq t_{now} - t' \leq w))\}$. Intuitively, assuming again that a timestamp is implemented as a set of periods, EX_w extends the end time of each period in a timestamp by w time units, and then merges adjacent periods together if they become overlapped after extension. For our running example, $\mathsf{EX}_2(R)$ is shown in Figure 5 and $\mathsf{EX}_2(R \cup \triangle R)$ in Figure 7. The fact that they happen to be equal explains why $\Pi_{\{\texttt{count}(*)\},2}(R)$ is unaffected by $\triangle R$.

Now, the problem of propagating changes through $\Pi_{A',w}$ can be reduced to first propagating changes through EX_w, and then through $\Pi_{A',0}$. Change propagation equations for EX_w are:

$$\begin{aligned}
\mathsf{EX}_w(R \Leftrightarrow \bigtriangledown R) &= \mathsf{EX}_w(R) \Leftrightarrow \Big(\mathsf{EX}_w(\bigtriangledown R) \Leftrightarrow \mathsf{EX}_w(R \Leftrightarrow \bigtriangledown R)\Big);\\
\mathsf{EX}_w(R \cup \triangle R) &= \mathsf{EX}_w(R) \cup \Big(\mathsf{EX}_w(\triangle R) \Leftrightarrow \mathsf{EX}_w(R)\Big).
\end{aligned}$$

To illustrate the reduction process, take insertion as an example:

$$\Pi_{A',w}(R \cup \triangle R) = \Pi_{A',0}(\mathsf{EX}_w(R \cup \triangle R)) = \Pi_{A',0}\Big(\mathsf{EX}_w(R) \cup (\mathsf{EX}_w(\triangle R) \Leftrightarrow \mathsf{EX}_w(R))\Big).$$

Applying (c.18), we obtain a maintenance expression that references $\Pi_{A',0}(\mathsf{EX}_w(R))$, which is exactly the same as $\Pi_{A',w}(R)$, the materialized view we are maintaining. Therefore, it is unnecessary to access R except for computing $\mathsf{EX}_w(\triangle R) - \mathsf{EX}_w(R)$. In practice, since there are usually constraints on the timestamp of $\triangle R$, we can avoid keeping the entire contents of R. Propagating deletions is similar. The change propagation equations are presented below. Note that we aggregate $\triangle R'$ and $\bigtriangledown R'$ using $\Pi_{A',0}$ rather than $\Pi_{A',w}$.

Theorem 5.2 In the following, we require that $\bigtriangledown R - R = \varnothing$.

(c.19) $\Pi_{A',w}(R \Leftrightarrow \bigtriangledown R) = \Pi_{A',w}(R) \Leftrightarrow \pi_{\$2.A}(J) \cup \pi_{\{g(\$2.a,\$1.a) \mid a \in A'\}}(\sigma_{\$2.\texttt{count}(*)-\$1.\texttt{count}(*)>0}(J))$,
where $\bigtriangledown R' = \mathsf{EX}_w(\bigtriangledown R) \Leftrightarrow \mathsf{EX}_w(R \Leftrightarrow \bigtriangledown R)$
and $J = \bowtie_{\$1.GB(A')=\$2.GB(A')}(\Pi_{A',0}(\bigtriangledown R'),\ \Pi_{A',w}(R))$

(c.20) $\Pi_{A',w}(R \cup \triangle R) = \Pi_{A',w}(R) \Leftrightarrow \pi_{\$2.A}(J) \cup \pi_{\{f(\$2.a,\$1.a) \mid a \in A'\}}(J) \cup (\Pi_{A',0}(\triangle R') \Leftrightarrow \pi_{\$1.A}(J))$,
where $\triangle R' = \mathsf{EX}_w(\triangle R) \Leftrightarrow \mathsf{EX}_w(R)$
and $J = \bowtie_{\$1.GB(A')=\$2.GB(A')}(\Pi_{A',0}(\triangle R'),\ \Pi_{A',w}(R))$

Proof outline: Reduction to $\Pi_{A',0}$ using $\Pi_{A',w}(R) = \Pi_{A',0}(\mathsf{EX}_w(R))$. Then follows from the change propagation equations for EX_w and Theorem 5.1. $\square$

Finally, we turn to the problem of refreshing cumulative aggregate views with respect to the current time. Unlike the τ-reducible instantaneous aggregates, the value of a cumulative aggregate at time t_{now} may not stay the same as time advances, even if there are no updates. In our running example, the value of $\Pi_{A',2}(R)$ at $t_{now} = 97$ is 2. This count includes tuple a because a is valid at some point in the window $[t_{now} - 2, t_{now}]$. As time advances, the window moves to the right, but the valid period of a never grows since `NOW` is not contained in this period. Eventually, a is no longer valid in the window and the count will decrease. Therefore, when refreshing a cumulative aggregate last refreshed at time t, we need to compute the aggregate for the period $[t + 1, t_{now}]$.

Theorem 5.3 In the following, $\mathsf{FR}_d(R)$ returns the fragment of R in the period d. It can be defined as a shorthand for $\pi_{\$1.A}(\bowtie\!\!\lhd_{\mathtt{true}}(R, D))$, where D is a temporal relation containing a single tuple with timestamp $\{d\}$.

$$
\begin{aligned}
\text{(r.9)}\quad \Pi_{A',w}(\psi_t([\mathcal{E}]_t)) = {} & \psi_t([\Pi_{A',w}(\mathcal{E})]_t) \\
& \Leftrightarrow \mathsf{FR}_{[t+1,\mathtt{NOW}]}\Big(\sigma_{\mathtt{NOW}\in\mathtt{T}}\Big(\psi_t([\Pi_{A',w}(\mathcal{E})]_t)\Big)\Big) \\
& \cup \mathsf{FR}_{[t+1,\mathtt{NOW}]}\Big(\Pi_{A',w}(\mathsf{FR}_{[t+1-w,\mathtt{NOW}]}(\psi_t([\mathcal{E}]_t)))\Big)
\end{aligned}
$$

Proof outline: Follows from the definition of $\Pi_{A',w}$. □

Intuitively, for all tuples valid at `NOW` in the materialized result of the aggregate, we terminate their timestamps at t. Then, we compute the new values of the aggregate for the period $[t + 1, \mathtt{NOW}]$ using the contents of $\mathcal{E}$ during $[t + 1 - w, \mathtt{NOW}]$, and append them to the result.

6 Implementation

In this section we describe how our temporal view maintenance framework is being integrated into *WHIPS* (*WareHouse Information Prototype at Stanford*). For details on the basic (non-temporal) WHIPS system, see [WGL+96]. The overall architecture of the temporal WHIPS system is illustrated in Figure 8.

For each temporal view exported by the warehouse, there is a *temporal view manager* responsible for maintaining the view. In contrast to non-temporal WHIPS, the concept of *self-maintainability* [BCL89, GJM96, QGMW96] is essential for temporal views since temporal base relations are not available at non-temporal sources: In general, each temporal view manager must also maintain a set of *auxiliary views* so that the original view together with the auxiliary views can be maintained without sending queries back to the sources, exactly as in [QGMW96]. In the worst case, the auxiliary views would be the entire temporal base relations, but as we have seen, many views do not require these relations to be materialized fully. Currently, view managers in our prototype maintain their own sets of auxiliary views independently, without sharing or interaction, although sharing is an important optimization, as discussed below. All materialized views, including auxiliary ones, are stored in the *temporal warehouse* component. The warehouse can be any temporal DBMS with TSQL2 support, such as *TimeCenter* [B+]. We are currently using our "home grown" temporal DBMS called *TIPS*.

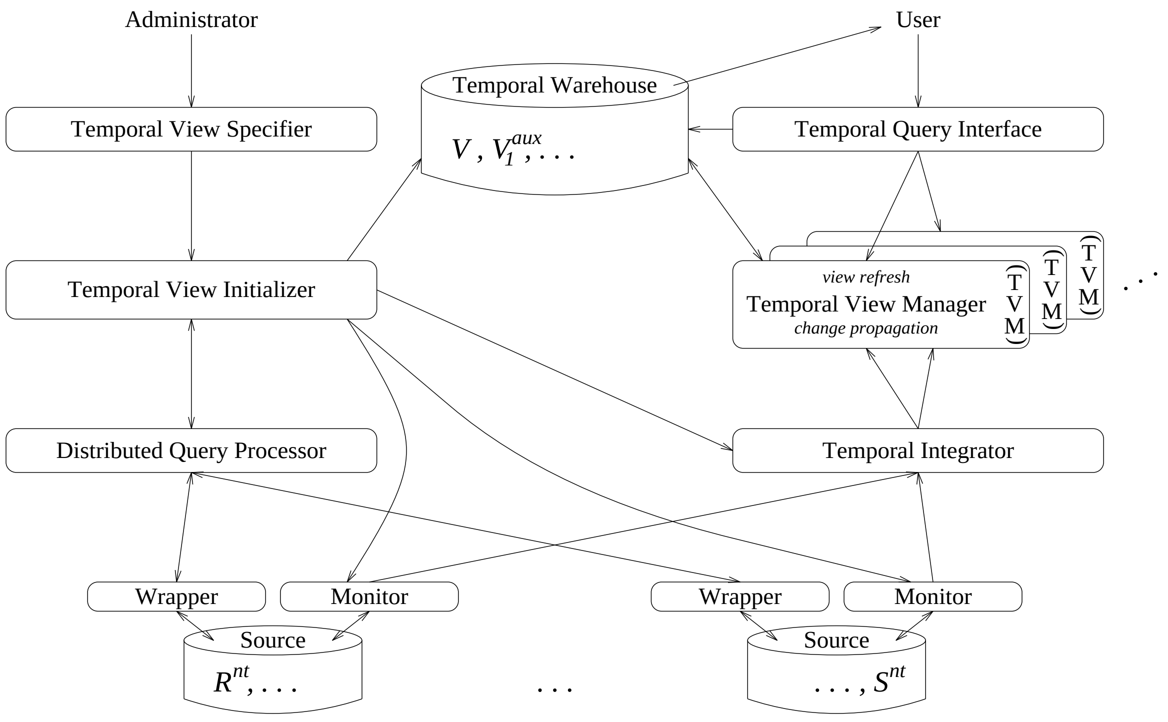


Figure 8: Temporal WHIPS architecture.

A view is defined by warehouse administrators using the *temporal view specifier*, which then passes the definition to the *temporal view initializer*. As the first step, the view initializer chooses an appropriate set of auxiliary views to maintain together with the new view, and creates schemas for these tables at the temporal warehouse. Currently we simply materialize relevant portions of all temporal base relations referenced in the view in addition to the view itself. Second, the view initializer asks the *temporal integrator* to spawn a new view manager, and sets up *monitors* to detect relevant source updates [Wid95, WGL+96]. Finally, the view initializer initializes the contents of the newly created tables at the warehouse by issuing queries to the *distributed query processor*. The query processor contacts the *wrapper* for each source involved in the view. For the purpose of initializing views, we treat the current snapshots of source relations at non-temporal sources as insertions at t_{now}; this issue will be revisited below.

As discussed throughout this paper, temporal views need to be maintained when source relations are updated and when time advances. Source updates are detected by monitors and sent to the integrator, exactly as in non-temporal warehousing systems [WGL+96]. The temporal integrator first converts the non-temporal source updates to updates to temporal base relations, following the procedure described in Section 4.1. It then forwards these updates to all relevant view managers. The view managers use change propagation (Section 4.2) and view refresh (Section 4.3) procedures to update the materialized views in the warehouse. When a user query is presented to the warehouse, it is intercepted by the *temporal query interface*. For each view referenced in the query, the query interface first asks the corresponding view manager to refresh the view, since the view may have become outdated as time advanced. After all relevant views have been refreshed, the warehouse answers the user query.

Some challenging problems have been encountered in the implementation, as described below. We plan to investigate these problems further; meanwhile, the prototype implements simple but functional solutions.

- As explained above, self-maintainability of temporal views is required when sources are non-temporal. Moreover, because views on history tend to become very large, it is important to enable auxiliary views to be shared among different view managers. How to choose auxiliary views for a set of temporal views so that together they all can be efficiently self-maintained is an open problem. [QGMW96] considers self-maintainability for a single, non-temporal view only. [RSS96] presents a framework for choosing auxiliary views that can be extended to a set of non-temporal views, but the criteria is efficiency, not self-maintainability.

- Initializing the contents of a temporal view can be tricky. In particular, we need to assign "sensible" timestamps to tuples from non-temporal sources. Currently we simply treat initial source tuples as newly inserted. However, for some of these tuples, part of their history may have already been recorded by the warehouse in other views. If we ignore this information, multiple views may reflect inconsistent source histories. How to use existing partial temporal information to initialize new views in order to avoid inconsistency presents another interesting problem.

7 Conclusion and Future Work

Providing temporal views over non-temporal source data is a very useful feature of a data warehouse. In this paper, we have presented a systematic approach to maintaining temporal views over non-temporal information sources in a data warehousing environment. We have identified the two cases in which a materialized temporal view needs to be updated: when base relations change, and when time advances. While the former case is similar to the non-temporal view maintenance problem, the latter is unique to temporal views. We have provided incremental techniques to deal with both cases. In order to avoid "reinventing the wheel", we have reused known relational equalities as much as possible in our change propagation equations, by exploiting the relationship between τ-reducible and θ-reducible operators and their non-temporal counterparts, and by introducing the EX_w operator for temporal aggregates. Our framework is useful not only in data warehousing, but also in other settings where the temporal view maintenance problem arises.

As described in Section 6, we are currently in the process of implementing our algorithms within the WHIPS system. We plan to evaluate the efficiency of the maintenance expressions by measuring their run-time performance and the amount of auxiliary storage required to make temporal views self-maintainable. Another possible enhancement to our framework is to allow different degrees of "laziness" or "eagerness" in propagating source updates and refreshing views. This flexibility will complicate the algorithm, but it may yield better performance because multiple updates can be combined in batch and processed more efficiently. Finally, when we incorporate timestamps from the sources, retroactive updates can be introduced, changing the warehouse's knowledge about past history. While our maintenance algorithm still works if we simply regard warehouse views as *valid time relations*, we may consider taking a more general approach in which each update has two timestamps: the actual time of the update at the source, and the time when the warehouse receives the update. Then, a temporal warehouse view may be modeled by a *bitemporal relation*, where the valid time dimension reflects the history of the base relations as observed by sources, and the transaction time dimension records the history of the warehouse's knowledge.

References

[B$^+$] M. H. Böhlen et al. TimeCenter homepage. Available on web at: `http://www.iesd.auc.dk/general/DBS/tdb/TimeCenter/`.

[BCL89] J. A. Blakeley, N. Coburn, and P.-Å. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions 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 *Proceedings of the International Workshop on Temporal Databases*, pages 153–174, September 1995.

[BL89] M. Bassiouni and M. Llewellyn. On the definition and maintenance of database views with time-varying domains. In *Proceedings of the 13th Annual International Computer Software and Applications Conference*, pages 201–208, September 1989.

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

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

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

[CT95] J. Chomicki and D. Toman. Implementing temporal integrity constraints using an active DBMS. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):566–581, August 1995.

[GHQ95] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proceedings of the 1995 International Conference on Very Large Data Bases*, pages 358–369, September 1995.

[GJM96] A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views. In *Proceedings of the 1996 International Conference on Extending Database Technology*, March 1996.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proceedings of the 1995 ACM SIGMOD International Conference 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*, 18(2):3–18, June 1995.

[JM93] C. S. Jensen and L. Mark. Differential query processing in transaction-time databases. In *Temporal Databases: Theory, Design, and Implementation*, chapter 19, pages 457–491. Benjamin/Cummings, 1993.

[JMR91] C. S. Jensen, L. Mark, and N. Roussopoulos. Incremental implementation model for relational databases with transaction time. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, December 1991.

[JMS95] H. V. Jagadish, I. S. Mumick, and A. Silberschatz. View maintenance issues for the chronicle data model. In *Proceedings of the 1995 ACM Symposium 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.

[LGM97] W. Labio and H. Garcia-Molina. Expiring data from the warehouse. Technical report, Stanford University, February 1997. Available on web as:
`http://www-db.stanford.edu/pub/papers/expire.ps`.

[LW95] D. Lomet and J. Widom, editors. *Special Issue on Materialized Views and Data Warehousing, IEEE Data Engineering Bulletin,* 18(2), June 1995.

[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 Transactions on Knowledge and Data Engineering*, 7(4):513–532, August 1995.

[Ple93] D. Plexousakis. Integrity constraint and rule maintenance in temporal deductive knowledge bases. In *Proceedings of the 1993 International Conference on Very Large Data Bases*, pages 146–157, August 1993.

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

[Qua96] D. Quass. Maintenance expressions for views with aggregation. In *Proceedings of the 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 Transactions on Knowledge and Data Engineering*, 3(3):337–341, September 1991.

[RSS96] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 447–458, June 1996.

[SGM93] R. T. Snodgrass, S. Gomez, and L. E. McKenzie. Aggregates in the temporal query language TQuel. *IEEE Transactions on Knowledge and Data Engineering*, 5(5):826–842, October 1993.

[SJS95] M. D. Soo, C. S. Jensen, and R. T. Snodgrass. An algebra for TSQL2. In *The TSQL2 Temporal Query Language*, pages 505–546. Kluwer Academic Press, September 1995.

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

[WGL$^+$96] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. A system prototype for warehouse view maintenance. In *Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications*, pages 26–33, June 1996.

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

[YW98] J. Yang and J. Widom. Maintaining temporal views over non-temporal information sources for data warehousing. In *Proceedings of the 1998 International Conference on Extending Database Technology*, pages 389–403, March 1998.

A Proofs of Theorems

Proof of Theorem 3.1. Follows from the definition for τ and Definition 3.1. It is obvious that $R = S$ implies $\forall t, t_0 \le t \le t_{now} : \tau_t(R) =_{nt} \tau_t(S)$. We now show that the converse is also true. Because of symmetry, it suffices to show $\forall t, t_0 \le t \le t_{now} : \tau_t(R) =_{nt} \tau_t(S)$ implies that any tuple in R is also in S. Suppose that $\langle u, c\rangle \in R$. By Definition 3.1, there exists $t \in c$ such that $t_0 \le t \le t_{now}$. Thus $u \in \tau_t(R) = \tau_t(S)$, *i.e.*, there exists a tuple $\langle u, c'\rangle \in S$. Furthermore, this tuple is unique because the explicit attributes include a key. For any $t \in \mathbb{T}$, $t \in c \iff t_0 \le t \le t_{now} \wedge u \in \tau_t(R) \iff t_0 \le t \le t_{now} \wedge u \in \tau_t(S) \iff t \in c'$. In the special case of NOW, $\texttt{NOW} \in c \iff t_{now} \in c \iff u \in \tau_{t_{now}}(R) \iff u \in \tau_{t_{now}}(S) \iff t_{now} \in c' \iff \texttt{NOW} \in c'$. Therefore, $c = c'$ and $\langle u, c\rangle \in S$. $\square$

Proof of Theorem 3.2. Induction on the structure of expressions. Follows from Theorem 3.1 and Definition 3.2. Let , $(\mathcal{E})$ denote the expression obtained by replacing, in $\mathcal{E}$, each τ-reducible operator by its non-temporal counterpart, and each temporal relation symbol R by $\tau_t(R)$. First we show $\tau_t(\mathcal{E}) =_{nt}$, $(\mathcal{E})$ by induction. The base case is trivial: $\tau_t(R) =_{nt}$, (R). Now suppose $\mathcal{E} = op(\mathcal{E}_1, ..., \mathcal{E}_n)$. $\tau_t(\mathcal{E}) =_{nt} \tau_t(op(\mathcal{E}_1, ..., \mathcal{E}_n)) =_{nt} op^{nt}(\tau_t(\mathcal{E}_1), ..., \tau_t(\mathcal{E}_n)) =_{nt} op^{nt}($, $(\mathcal{E}_1), ...,$, $(\mathcal{E}_n)) =_{nt}$, $(\mathcal{E})$, by Definition 3.2 and the inductive hypothesis. Applying this result to $\mathcal{E}$ and $\mathcal{F}$, we get $\tau_t(\mathcal{E}) =_{nt}$, $(\mathcal{E})$ and $\tau_t(\mathcal{F}) =_{nt}$, $(\mathcal{F})$. Since $\mathcal{E}^{nt} =_{nt} \mathcal{F}^{nt}$ holds in relational algebra, , $(\mathcal{E}) =_{nt}$, $(\mathcal{F})$, *i.e.*, $\tau_t(\mathcal{E}) =_{nt} \tau_t(\mathcal{F})$. By Theorem 3.1, $\mathcal{E} = \mathcal{F}$. $\square$

Proof of Theorem 3.3. Follows from the definitions of these operators. To show that $\tau_t(R \Leftrightarrow S) =_{nt} \tau_t(R) \Leftrightarrow^{nt} \tau_t(S)$:

$$\begin{aligned}
& u \in \tau_t(R \Leftrightarrow S) \\
\iff & \exists c : t \in c \wedge \langle u, c\rangle \in R \Leftrightarrow S \\
\iff & \exists r \in R : r.A_R = u \wedge t \in r.\texttt{T} \Leftrightarrow h(S, r, A_R) \\
\iff & \exists r \in R : r.A_R = u \wedge t \in r.\texttt{T} \wedge t \notin h(S, r, A_R) \\
\iff & u \in \tau_t(R) \wedge u \notin \tau_t(S) \\
\iff & u \in \tau_t(R) \Leftrightarrow^{nt} \tau_t(S).
\end{aligned}$$

To show that $\tau_t(R \cup S) =_{nt} \tau_t(R) \cup^{nt} \tau_t(S)$:

$$\begin{aligned}
& u \in \tau_t(R \cup S) \\
\iff & \exists c : t \in c \wedge \langle u, c\rangle \in R \cup S \\
\iff & (\exists r \in R : u = r.A_R \wedge t \in r.\texttt{T} \cup h(S, r, A_R)) \vee (\exists s \in S : u = s.A_R \wedge t \in s.\texttt{T} \cup h(R, s, A_R)) \\
\iff & (\exists r \in R : u = r.A_R \wedge t \in r.\texttt{T}) \vee (\exists r \in R : u = r.A_R \wedge t \in h(S, r, A_R))
\end{aligned}$$

$$\begin{aligned}
& \lor (\exists s \in S : u = s.A_R \land t \in s.\mathtt{T}) \lor (\exists s \in S : u = s.A_R \land t \in h(R, s, A_R)) \\
\Longleftrightarrow\ & (\exists r \in R : u = r.A_R \land t \in r.\mathtt{T}) \lor (\exists s \in S : u = s.A_R \land t \in s.\mathtt{T}) \\
\Longleftrightarrow\ & u \in \tau_t(R) \lor u \in \tau_t(S) \\
\Longleftrightarrow\ & u \in \tau_t(R) \cup^{nt} \tau_t(S).
\end{aligned}$$

To show that $\tau_t(\pi_{A'}(R)) =_{nt} \pi_{A'}^{nt}(\tau_t(R))$:

$$\begin{aligned}
& u \in \tau_t(\pi_{A'}(R)) \\
\Longleftrightarrow\ & \exists c : t \in c \land \langle u, c\rangle \in \pi_{A'}(R) \\
\Longleftrightarrow\ & \exists r \in R : u = r.A' \land t \in h(R, r, A') \\
\Longleftrightarrow\ & \exists r \in R : u = r.A' \land t \in r.\mathtt{T} \\
\Longleftrightarrow\ & u \in \pi_{A'}^{nt}(\tau_t(R)).
\end{aligned}$$

To show that $\tau_t(\sigma_p(R)) =_{nt} \sigma_p^{nt}(\tau_t(R))$, where p contains no references to `NOW` or `T`:

$$\begin{aligned}
& u \in \tau_t(\sigma_p(R)) \\
\Longleftrightarrow\ & \exists c : t \in c \land \langle u, c\rangle \in \sigma_p(R) \\
\Longleftrightarrow\ & \exists r \in R : u = r.A_R \land t \in r.\mathtt{T} \land p(u) \\
\Longleftrightarrow\ & u \in \tau_t(R) \land p(u) \\
\Longleftrightarrow\ & u \in \sigma_p^{nt}(\tau_t(R)).
\end{aligned}$$

To show that $\tau_t(\bowtie_p(R_1, ..., R_n)) =_{nt} \bowtie_p^{nt}(\tau_t(R_1), ...\tau_t(R_n))$, where p contains no references to `NOW` or `T`:

$$\begin{aligned}
& u \in \tau_t(\bowtie_p(R_1, ..., R_n)) \\
\Longleftrightarrow\ & \exists c : t \in c \land \langle u, c\rangle \in \bowtie_p(R_1, ..., R_n) \\
\Longleftrightarrow\ & \exists r_i \in R_i : u = \langle r_1.A_{R_1}, ..., r_n.A_{R_n}\rangle \land t \in r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \land p(r_1.A_{R_1}, ..., r_n.A_{R_n}) \\
\Longleftrightarrow\ & \exists r_i \in R_i : u = \langle r_1.A_{R_1}, ..., r_n.A_{R_n}\rangle \land t \in r_1.\mathtt{T} \land ... \land t \in r_n.\mathtt{T} \land p(r_1.A_{R_1}, ..., r_n.A_{R_n}) \\
\Longleftrightarrow\ & \exists r_i \in R_i : u = \langle r_1.A_{R_1}, ..., r_n.A_{R_n}\rangle \land r_1.A_{R_1} \in \tau_t(R_1) \land ... \land r_n.A_{R_n} \in \tau_t(R_n) \\
& \land p(r_1.A_{R_1}, ..., r_n.A_{R_n}) \\
\Longleftrightarrow\ & u \in \bowtie_p^{nt}(\tau_t(R_1), ...\tau_t(R_n)).
\end{aligned}$$

□

Proof of Theorem 3.4. Follows from the definition for θ and Definition 3.1. It is obvious that $R = S$ implies $\theta(R) =_{nt} \theta(S)$. We now show that the converse is also true: $\langle u, c\rangle \in R \Longleftrightarrow \langle u, c\rangle \in \theta(R) \Longleftrightarrow \langle u, c\rangle \in \theta(S) \Longleftrightarrow \langle u, c\rangle \in S$. □

Proof of Theorem 3.5. Follows from the definitions for θ and σ_p.

$$\begin{aligned}
& \langle u, c\rangle \in \theta(\sigma_p(R)) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \sigma_p(R) \\
\Longleftrightarrow\ & \langle u, c\rangle \in R \land p(u, c) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \theta(R) \land p(u, c) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \sigma_p^{nt}(\theta(R)).
\end{aligned}$$

□

Proof of Theorem 3.6. Follows from the definitions for θ and $\bowtie_p$.

$$\begin{array}{ll}
 & \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,R_n)) \\
\Longleftrightarrow & \langle u,c\rangle \in \bowtie_p(R_1,...,R_n) \\
\Longleftrightarrow & \exists r_i \in R_i : u = \langle r_1.A_{R_1},...,r_n.A_{R_n}\rangle \wedge c = r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \neq \varnothing \wedge p(r_1,...,r_n) \\
\Longleftrightarrow & \exists r_i \in \theta(R_i) : u = \langle r_1.A_{R_1},...,r_n.A_{R_n}\rangle \wedge c = r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \neq \varnothing \wedge p(r_1,...,r_n) \\
\Longleftrightarrow & \exists r_i \in \theta(R_i) : u = \langle r_1.A_{R_1},...,r_n.A_{R_n}\rangle \wedge c = r_1.\mathtt{T} \cap ... \cap r_n.\mathtt{T} \\
 & \wedge \langle r_1.A_{R_1},...,r_n.A_{R_n}, r_1.\mathtt{T},...,r_n.\mathtt{T}\rangle \in \bowtie_{p'}^{nt}(\theta(R_1),...,\theta(R_n)) \\
\Longleftrightarrow & \langle u,c\rangle \in \pi_{A'}^{nt}(\bowtie_{p'}^{nt}(\theta(R_1),...,\theta(R_n))).
\end{array}$$

□

Proof of Theorem 4.1. Follows from Definition 4.1 and the definitions of $\Leftrightarrow$ and $\cup$. Let R_{old}^{nt} denote the relation R^{nt} immediately before the update, and let R_{new}^{nt} denote the relation after the update. $R_{new}^{nt} =_{nt} (R_{old}^{nt} \Leftrightarrow^{nt} \bigtriangledown R^{nt}) \cup^{nt} \triangle R^{nt}$. Since R_{old} is a faithful history encoding of R^{nt} immediately before the update, we have: $\forall t, t_0 \leq t < t_{now}$: $\tau_t(R_{old})$ is identical to the state of R^{nt} at t, and $\tau_{t_{now}}(R_{old}) =_{nt} R_{old}^{nt}$. Now consider R_{new}. $\forall t, t_0 \leq t < t_{now} : \tau_t(R_{new}) =_{nt} \tau_t((R_{old} \Leftrightarrow \bigtriangledown R) \cup \triangle R) =_{nt} (\tau_t(R_{old}) \Leftrightarrow^{nt} \tau_t(\bigtriangledown R)) \cup^{nt} \tau_t(\triangle R) =_{nt} (\tau_t(R_{old}) \Leftrightarrow^{nt} \varnothing) \cup^{nt} \varnothing =_{nt} \tau_t(R_{old})$, which is identical to the state of R^{nt} at t. For $t = t_{now}$, $\tau_{t_{now}}(R_{new}) =_{nt} (\tau_{t_{now}}(R_{old}) \Leftrightarrow^{nt} \tau_{t_{now}}(\bigtriangledown R)) \cup^{nt} \tau_{t_{now}}(\triangle R) =_{nt} (R_{old}^{nt} \Leftrightarrow^{nt} \bigtriangledown R^{nt}) \cup^{nt} \triangle R^{nt} =_{nt} R_{new}^{nt}$. Therefore R_{new} is a faithful history encoding of R^{nt} after the update. □

Proof of Theorem 4.3. Follows from the definitions for θ, $\Leftrightarrow$, and $\cup$. To prove (i), we will use Equality (c.5) given in Theorem 4.2.

$$\begin{array}{ll}
 & (R \Leftrightarrow \bigtriangledown_2 R) \cup \triangle_2 R \\
= & (R \Leftrightarrow \bigtriangledown_2 R) \cup ((\bigtriangledown_2 R \Leftrightarrow \bigtriangledown_1 R) \cup \triangle_1 R) \\
= & ((R \Leftrightarrow \bigtriangledown_2 R) \cup (\bigtriangledown_2 R \Leftrightarrow \bigtriangledown_1 R)) \cup \triangle_1 R \\
= & (((R \Leftrightarrow \bigtriangledown_2 R) \cup \bigtriangledown_2 R) \Leftrightarrow (\bigtriangledown_1 R \Leftrightarrow (R \Leftrightarrow \bigtriangledown_2 R))) \cup \triangle_1 R \\
= & (R \Leftrightarrow (\bigtriangledown_1 R \Leftrightarrow (R \Leftrightarrow \bigtriangledown_2 R))) \cup \triangle_1 R.
\end{array}$$

Now we only need to show $\bigtriangledown_1 R \Leftrightarrow (R \Leftrightarrow \bigtriangledown_2 R) = \bigtriangledown_1 R$, so that the expression above can be reduced to $(R \Leftrightarrow \bigtriangledown_1 R) \cup \triangle_1 R = R_{new}$. To this end, note that $\bigtriangledown_1 R \Leftrightarrow (R \Leftrightarrow \bigtriangledown_2 R) = \{\langle u,c\rangle \mid \exists r \in \bigtriangledown_1 R : u = r.A_R \wedge c = r.\mathtt{T} \Leftrightarrow h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R) \neq \varnothing\}$. If $h(R, r, A_R) = \varnothing$, then obviously $h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R) = \varnothing$. If $h(R, r, A_R) \neq \varnothing$, $r \in \bigtriangledown_1 R$ implies $h(R, r, A_R) = h(\bigtriangledown_2 R, r, A_R)$, and since A_R includes a key, $h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R) = \varnothing$. In either case, $\bigtriangledown_1 R \Leftrightarrow (R \Leftrightarrow \bigtriangledown_2 R) = \{\langle u,c\rangle \mid \exists r \in \bigtriangledown_1 R : u = r.A_R \wedge c = r.\mathtt{T} \Leftrightarrow \varnothing\} = \bigtriangledown_1 R$.

To prove (ii):

$$\begin{array}{lll}
 & \langle u,c\rangle \in \theta(R \Leftrightarrow \bigtriangledown_2 R) & \\
\Longleftrightarrow & \langle u,c\rangle \in R \Leftrightarrow \bigtriangledown_2 R & \\
\Longleftrightarrow & \exists r \in R : u = r.A_R \wedge c = r.\mathtt{T} \Leftrightarrow h(\bigtriangledown_2 R, r, A_R) \neq \varnothing & (3) \\
\Longleftrightarrow & \exists r \in R : u = r.A_R \wedge c = r.\mathtt{T} \wedge h(\bigtriangledown_2 R, r, A_R) = \varnothing & (4) \\
\Longleftrightarrow & \langle u,c\rangle \in R \wedge \langle u,c\rangle \notin \bigtriangledown_2 R & (5) \\
\Longleftrightarrow & \langle u,c\rangle \in \theta(R) \wedge \langle u,c\rangle \notin \theta(\bigtriangledown_2 R) & \\
\Longleftrightarrow & \langle u,c\rangle \in \theta(R) \Leftrightarrow^{nt} \theta(\bigtriangledown_2 R). &
\end{array}$$

(3) $\Rightarrow$ (4) because $\forall r \in R : h(\bigtriangledown_2 R, r, A_R) = \varnothing \vee h(\bigtriangledown_2 R, r, A_R) = r.\mathtt{T}$. Since $r.\mathtt{T} \Leftrightarrow h(\bigtriangledown_2 R, r, A_R) \neq \varnothing$, it must be the case that $h(\bigtriangledown_2 R, r, A_R) = \varnothing$. (5) $\Rightarrow$ (4) can be shown as follows: Suppose that, on the contrary, $h(\bigtriangledown_2 R, r, A_R) \neq \varnothing$. Then $\exists c' : \langle u, c' \rangle \in \bigtriangledown_2 R$, and therefore, $\langle u, c' \rangle \in R$. Since the set of explicit attributes contains a key, $c = c'$ and $\langle u, c \rangle \in \bigtriangledown_2 R$, which is a contradiction.

To prove (iii):

$$
\begin{aligned}
& \langle u,c\rangle \in \theta((R \Leftrightarrow \bigtriangledown_2 R) \cup \triangle_2 R) \\
\Longleftrightarrow\quad & \langle u,c\rangle \in (R \Leftrightarrow \bigtriangledown_2 R) \cup \triangle_2 R \\
\Longleftrightarrow\quad & (\exists r \in R \Leftrightarrow \bigtriangledown_2 R : u = r.A_R \wedge c = r.\mathtt{T} \cup h(\triangle_2 R, r, A_R)) \\
& \vee (\exists r \in \triangle_2 R : u = r.A_R \wedge c = r.\mathtt{T} \cup h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R)) && (6) \\
\Longleftrightarrow\quad & (\exists r \in R \Leftrightarrow \bigtriangledown_2 R : u = r.A_R \wedge c = r.\mathtt{T}) \vee (\exists r \in \triangle_2 R : u = r.A_R \wedge c = r.\mathtt{T}) && (7) \\
\Longleftrightarrow\quad & \langle u,c\rangle \in R \Leftrightarrow \bigtriangledown_2 R \vee \langle u,c\rangle \in \triangle_2 R \\
\Longleftrightarrow\quad & \langle u,c\rangle \in \theta(R \Leftrightarrow \bigtriangledown_2 R) \vee \langle u,c\rangle \in \theta(\triangle_2 R) \\
\Longleftrightarrow\quad & \langle u,c\rangle \in \theta(R \Leftrightarrow \bigtriangledown_2 R) \cup^{nt} \theta(\triangle_2 R).
\end{aligned}
$$

(6) $\Leftrightarrow$ (7) because (a) $r \in R \Leftrightarrow \bigtriangledown_2 R \Longrightarrow h(\triangle_2 R, r, A_R) = \varnothing$, and (b) $r \in \triangle_2 R \Longrightarrow h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R) = \varnothing$. We prove (a) first: $r \in R \Leftrightarrow \bigtriangledown_2 R \Longrightarrow r \in R \wedge h(\bigtriangledown_2 R, r, A_R) = \varnothing$, by the proof of (ii). Furthermore, $h(\triangle_1 R, r, A_R) = \varnothing$ because otherwise, $r \in R \wedge h(\triangle_1 R, r, A_R) \neq \varnothing \Longrightarrow h(\bigtriangledown_2 R, r, A_R) \neq \varnothing$, a contradiction. Since $\triangle_2 R = (\bigtriangledown_2 R \Leftrightarrow \bigtriangledown_1 R) \cup \triangle_1 R$, it is easy to see that $h(\bigtriangledown_2 R, r, A_R) = \varnothing \wedge h(\triangle_1 R, r, A_R) = \varnothing$ implies $h(\triangle_2 R, r, A_R) = \varnothing$. To prove (b), suppose that, on the contrary, $h(R \Leftrightarrow \bigtriangledown_2 R, r, A_R) \neq \varnothing$. Then $\exists r' \in R \Leftrightarrow \bigtriangledown_2 R : r'.A_R = r.A_R$. By (a), $r' \in R \Leftrightarrow \bigtriangledown_2 R \Longrightarrow h(\triangle_2 R, r', A_R) = \varnothing$. On the other hand, $h(\triangle_2 R, r, A_R) = h(\triangle_2 R, r', A_R) = \varnothing$. This contradicts the fact that $r \in \triangle_2 R$. □

In order to prove Theorem 4.4, we introduce the following two lemmas:

Lemma A.1 If $\theta(R \Leftrightarrow \bigtriangledown R) =_{nt} \theta(R) \Leftrightarrow^{nt} \theta(\bigtriangledown R)$, then $\forall r \in R : r.\mathtt{T} = h(\bigtriangledown R, r, A_R) \vee r.\mathtt{T} \Leftrightarrow h(\bigtriangledown R, r, A_R) = r.\mathtt{T}$.

Proof: Suppose that, on the contrary, $\exists r \in R : r.\mathtt{T} \neq h(\bigtriangledown R, r, A_R) \wedge r.\mathtt{T} \Leftrightarrow h(\bigtriangledown R, r, A_R) \neq r.\mathtt{T}$. Since $r.\mathtt{T} \Leftrightarrow h(\bigtriangledown R, r, A_R) \neq r.\mathtt{T}$, $h(\bigtriangledown R, r, A_R) \neq \varnothing$, *i.e.*, $\exists r' \in \bigtriangledown R : r'.A_R = r.A_R$. Because A_R includes a key, $r'.\mathtt{T} = h(\bigtriangledown R, r, A_R) \neq r.\mathtt{T}$; therefore $r \notin \bigtriangledown R$. $\langle r.A_R, r.\mathtt{T}\rangle \in \theta(R) \wedge \langle r.A_R, r.\mathtt{T}\rangle \notin \theta(\bigtriangledown R)$ implies that $\langle r.A_R, r.\mathtt{T}\rangle \in \theta(R) \Leftrightarrow^{nt} \theta(\bigtriangledown R)$. On the other hand, $\langle r.A_R, r.\mathtt{T}\rangle \notin \theta(R \Leftrightarrow \bigtriangledown R)$ because $r.\mathtt{T} \Leftrightarrow h(\bigtriangledown R, r, A_R) \neq r.\mathtt{T}$. This contradicts the fact that $\theta(R \Leftrightarrow \bigtriangledown R) =_{nt} \theta(R) \Leftrightarrow^{nt} \theta(\bigtriangledown R)$. □

Lemma A.2 If $\theta(R \cup \triangle R) =_{nt} \theta(R) \cup^{nt} \theta(\triangle R)$, then $\forall r \in R : r.\mathtt{T} \cup h(\triangle R, r, A_R) = r.\mathtt{T}$, and $\forall r \in \triangle R : r.\mathtt{T} \cup h(R, r, A_R) = r.\mathtt{T}$.

Proof: Because of symmetry, it suffices to show that $\forall r \in R : r.\mathtt{T} \cup h(\triangle R, r, A_R) = r.\mathtt{T}$. Suppose that, on the contrary, $\exists r \in R : r.\mathtt{T} \cup h(\triangle R, r, A_R) \neq r.\mathtt{T}$. Then, $\langle r.A_R, r.\mathtt{T}\rangle \notin \theta(R \cup \triangle R)$. On the other hand, $\langle r.A_R, r.\mathtt{T}\rangle \in \theta(R)$ implies that $\langle r.A_R, r.\mathtt{T}\rangle \in \theta(R) \cup^{nt} \theta(\triangle R)$. This contradicts the fact that $\theta(R \cup \triangle R) =_{nt} \theta(R) \cup^{nt} \theta(\triangle R)$. □

Proof of Theorem 4.4. Follows from Theorems 3.4, 3.5, 3.6 and the correctness of known equalities for non-temporal relational algebra. We will also make use of Lemmas A.1 and A.2 given above. By Theorem 3.4, in order to prove (c.13), it suffices to show that $\theta(\sigma_p(R \Leftrightarrow \bigtriangledown R)) =_{nt} \theta(\sigma_p(R) \Leftrightarrow \sigma_p(\bigtriangledown R))$:

$$
\begin{aligned}
& \theta(\sigma_p(R \Leftrightarrow \bigtriangledown R)) \\
=_{nt}\quad & \sigma_p^{nt}(\theta(R \Leftrightarrow \bigtriangledown R))
\end{aligned}
$$

$$\begin{aligned}
&=_{nt} \quad \sigma_p^{nt}(\theta(R) \Leftrightarrow^{nt} \theta(\nabla R)) \\
&=_{nt} \quad \sigma_p^{nt}(\theta(R)) \Leftrightarrow^{nt} \sigma_p^{nt}(\theta(\nabla R)) \\
&=_{nt} \quad \theta(\sigma_p(R)) \Leftrightarrow^{nt} \theta(\sigma_p(\nabla R)).
\end{aligned}$$

It remains to be shown that $\theta(\sigma_p(R)) \Leftrightarrow^{nt} \theta(\sigma_p(\nabla R)) =_{nt} \theta(\sigma_p(R) \Leftrightarrow \sigma_p(\nabla R))$. We shall first prove the following result:

$$\begin{aligned}
&\theta(R \Leftrightarrow \nabla R) =_{nt} \theta(R) \Leftrightarrow^{nt} \theta(\nabla R) \\
\Longrightarrow \quad &\forall r \in \sigma_p(R) : r.\mathtt{T} = h(\sigma_p(\nabla R), r, A_R) \vee r.\mathtt{T} \Leftrightarrow h(\sigma_p(\nabla R), r, A_R) = r.\mathtt{T}.
\end{aligned}$$

This result follows from Lemma A.1: $r \in \sigma_p(R) \Longrightarrow r \in R \Longrightarrow r.\mathtt{T} = h(\nabla R, r, A_R) \vee r.\mathtt{T} \Leftrightarrow h(\nabla R, r, A_R) = r.\mathtt{T}$. Note that $h(\sigma_p(\nabla R), r, A_R) = h(\nabla R, r, A_R)$ or $\varnothing$; it is easy to verify the above result for either case. Using this result, we can now show that $\theta(\sigma_p(R)) \Leftrightarrow^{nt} \theta(\sigma_p(\nabla R)) =_{nt} \theta(\sigma_p(R) \Leftrightarrow \sigma_p(\nabla R))$:

$$\begin{aligned}
&\langle u, c\rangle \in \theta(\sigma_p(R)) \Leftrightarrow^{nt} \theta(\sigma_p(\nabla R)) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \theta(\sigma_p(R)) \wedge \langle u, c\rangle \notin \theta(\sigma_p(\nabla R)) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \sigma_p(R) \wedge \langle u, c\rangle \notin \sigma_p(\nabla R) \\
\Longleftrightarrow \quad &\exists r \in \sigma_p(R) : u = r.A_R \wedge c = r.\mathtt{T} \wedge r.\mathtt{T} \Leftrightarrow h(\sigma_p(\nabla R), r, A_R) = r.\mathtt{T} \\
\Longleftrightarrow \quad &\exists r \in \sigma_p(R) : u = r.A_R \wedge c = r.\mathtt{T} \Leftrightarrow h(\sigma_p(\nabla R), r, A_R) \neq \varnothing \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \sigma_p(R) \Leftrightarrow \sigma_p(\nabla R) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \theta(\sigma_p(R) \Leftrightarrow \sigma_p(\nabla R)).
\end{aligned}$$

To prove (c.14), it suffices to show that $\theta(\sigma_p(R \cup \triangle R)) =_{nt} \theta(\sigma_p(R) \cup \sigma_p(\triangle R))$:

$$\begin{aligned}
& \quad \theta(\sigma_p(R \cup \triangle R)) \\
&=_{nt} \quad \sigma_p^{nt}(\theta(R \cup \triangle R)) \\
&=_{nt} \quad \sigma_p^{nt}(\theta(R) \cup^{nt} \theta(\triangle R)) \\
&=_{nt} \quad \sigma_p^{nt}(\theta(R)) \cup^{nt} \sigma_p^{nt}(\theta(\triangle R)) \\
&=_{nt} \quad \theta(\sigma_p(R)) \cup^{nt} \theta(\sigma_p(\triangle R)).
\end{aligned}$$

Now we need to show that $\theta(\sigma_p(R)) \cup^{nt} \theta(\sigma_p(\triangle R)) =_{nt} \theta(\sigma_p(R) \cup \sigma_p(\triangle R))$. To this end, observe that:

$$\begin{aligned}
&\theta(R \cup \triangle R) =_{nt} \theta(R) \cup^{nt} \theta(\triangle R) \\
\Longrightarrow \quad &(\forall r \in \sigma_p(R) : r.\mathtt{T} \cup h(\sigma_p(\triangle R), r, R_A) = r.\mathtt{T}) \\
&\wedge (\forall r \in \sigma_p(\triangle R) : r.\mathtt{T} \cup h(\sigma_p(R), r, R_A) = r.\mathtt{T}).
\end{aligned}$$

This follows from Lemma A.2, because any tuple in $\sigma_p(R)$ (or $\sigma_p(\triangle R)$) must also be in R (or $\triangle R$, respectively). Using this result, we have:

$$\begin{aligned}
&\langle u, c\rangle \in \theta(\sigma_p(R)) \cup^{nt} \theta(\sigma_p(\triangle R)) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \theta(\sigma_p(R)) \vee \langle u, c\rangle \in \theta(\sigma_p(\triangle R)) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \sigma_p(R) \vee \langle u, c\rangle \in \sigma_p(\triangle R) \\
\Longleftrightarrow \quad &(\exists r \in \sigma_p(R) : u = r.A_R \wedge c = r.\mathtt{T}) \vee (\exists r \in \sigma_p(\triangle R) : u = r.A_R \wedge c = r.\mathtt{T}) \\
\Longleftrightarrow \quad &(\exists r \in \sigma_p(R) : u = r.A_R \wedge c = r.\mathtt{T} \cup h(\sigma_p(\triangle R), r, A_R)) \\
&\vee (\exists r \in \sigma_p(\triangle R) : u = r.A_R \wedge c = r.\mathtt{T} \cup h(\sigma_p(R), r, A_R)) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \sigma_p(R) \cup \sigma_p(\triangle R) \\
\Longleftrightarrow \quad &\langle u, c\rangle \in \theta(\sigma_p(R) \cup \sigma_p(\triangle R)).
\end{aligned}$$

To prove (c.15), it suffices to show that $\theta(\bowtie_p(R_1, ..., R_i \Leftrightarrow \nabla R_i, ..., R_n)) =_{nt} \theta(\bowtie_p(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bowtie_p(R_1, ..., \nabla R_i, ..., R_n))$:

$$\begin{aligned}
& \theta(\bowtie_p(R_1, ..., R_i \Leftrightarrow \nabla R_i, ..., R_n)) \\
=_{nt}\ & \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i \Leftrightarrow \nabla R_i), ..., \theta(R_n))) \\
=_{nt}\ & \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i) \Leftrightarrow^{nt} \theta(\nabla R_i), ..., \theta(R_n))) \\
=_{nt}\ & \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n)) \Leftrightarrow^{nt} \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))) && (8) \\
=_{nt}\ & \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n))) \Leftrightarrow^{nt} \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))) && (9) \\
=_{nt}\ & \theta(\bowtie_p(R_1, ..., R_i, ..., R_n)) \Leftrightarrow^{nt} \theta(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n)) && (10) \\
=_{nt}\ & \theta(\bowtie_p(R_1, ..., R_i, ..., R_n) \Leftrightarrow \bowtie_p(R_1, ..., \nabla R_i, ..., R_n)). && (11)
\end{aligned}$$

We shall prove (8) $=_{nt}$ (9) first, and (10) $=_{nt}$ (11) later:

$$\begin{aligned}
& \langle u, c\rangle \in \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n)) \Leftrightarrow^{nt} \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))) \\
\Longleftrightarrow\ & \exists\langle r_1, ..., r_i, ..., r_n\rangle \in \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n)) \Leftrightarrow^{nt} \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n)) : \\
& u = \langle r.A_{R_1}, ..., r_i.A_{R_i}, ..., r_n.A_{R_n}\rangle \wedge c = r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T} \\
\Longleftrightarrow\ & \exists\langle r_1, ..., r_i, ..., r_n\rangle \in \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n)) : \\
& \langle r_1, ..., r_i, ..., r_n\rangle \notin \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n)) \\
& \wedge u = \langle r.A_{R_1}, ..., r_i.A_{R_i}, ..., r_n.A_{R_n}\rangle \wedge c = r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T} && (12) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n))) \\
& \wedge \langle u, c\rangle \notin \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))) && (13) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(R_i), ..., \theta(R_n))) \Leftrightarrow^{nt} (\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))).
\end{aligned}$$

Here, (12) $\Rightarrow$ (13) can be proven by contradiction. Suppose $\langle u, c\rangle \in \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n)))$. Then $\exists r'_i \in \nabla R_i : r'_i.A_{R_i} = r_i.A_{R_i} \wedge c = r_1.\mathrm{T} \cap ... \cap r'_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}$. On the other hand, $c = r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}$. Thus we have $\varnothing = (r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}) \Leftrightarrow (r_1.\mathrm{T} \cap ... \cap r'_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}) = r_1.\mathrm{T} \cap ... \cap (r_i \Leftrightarrow r'_i) \cap ... \cap r_n.\mathrm{T}$. By Lemma A.1, $r_i.\mathrm{T} = r'_i.\mathrm{T} \vee r_i.\mathrm{T} \Leftrightarrow r'_i.\mathrm{T} = r_i.\mathrm{T}$. Furthermore, since $\langle r_1, ..., r_i, ..., r_n\rangle \notin \bowtie^{nt}_{p'}(\theta(R_1), ..., \theta(\nabla R_i), ..., \theta(R_n))$, $r_i.\mathrm{T} \neq r'_i.\mathrm{T}$. Therefore $\varnothing = r_1.\mathrm{T} \cap ... \cap (r_i \Leftrightarrow r'_i) \cap ... \cap r_n.\mathrm{T} = r_1.\mathrm{T} \cap ... \cap r_i \cap ... \cap r_n.\mathrm{T} = c$, which is a contradiction.

Let A_R denote $A_{R_1} \cup ... \cup A_{R_n}$. To prove (10) $=_{nt}$ (11), we will use the following fact:

$$\begin{aligned}
& \theta(R_i \Leftrightarrow \nabla R_i) =_{nt} \theta(R_i) \Leftrightarrow^{nt} \theta(\nabla R_i) \\
\Longrightarrow\ & \forall r \in \bowtie_p(R_1, ..., R_i, ..., R_n) : \\
& r.\mathrm{T} = h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R) \vee r.\mathrm{T} \Leftrightarrow h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R) = r.\mathrm{T}.
\end{aligned}$$

This fact can be proven as follows: If $h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R) = \varnothing$, the fact obviously holds. Otherwise, let r be the result of temporally joining together $r_1 \in R_1, ..., r_i \in R_i, ..., r_n \in R_n$. Then $\exists r'_i \in \nabla R_i : r'_i.A_{R_i} = r_i.A_R \wedge h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R) = r_1.\mathrm{T} \cap ... \cap r'_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}$. By Lemma A.1, $r_i.\mathrm{T} = r'_i.\mathrm{T} \vee r_i.\mathrm{T} \Leftrightarrow r'_i.\mathrm{T} = r_i.\mathrm{T}$. If $r_i.\mathrm{T} = r'_i.\mathrm{T}$, then $r.\mathrm{T} = h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R)$. If $r_i.\mathrm{T} \Leftrightarrow r'_i.\mathrm{T} = r_i.\mathrm{T}$, then $r.\mathrm{T} \Leftrightarrow h(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n), r, A_R) = (r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}) \Leftrightarrow (r_1.\mathrm{T} \cap ... \cap r'_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T}) = r_1.\mathrm{T} \cap ... \cap (r_i.\mathrm{T} \Leftrightarrow r'_i.\mathrm{T}) \cap ... \cap r_n.\mathrm{T} = r_1.\mathrm{T} \cap ... \cap r_i.\mathrm{T} \cap ... \cap r_n.\mathrm{T} = r.\mathrm{T}$. Using this result, we are now ready to show that (10) $=_{nt}$ (11):

$$\begin{aligned}
& \langle u, c\rangle \in \theta(\bowtie_p(R_1, ..., R_i, ..., R_n)) \Leftrightarrow^{nt} \theta(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n)) \\
\Longleftrightarrow\ & \langle u, c\rangle \in \theta(\bowtie_p(R_1, ..., R_i, ..., R_n)) \wedge \langle u, c\rangle \notin \theta(\bowtie_p(R_1, ..., \nabla R_i, ..., R_n))
\end{aligned}$$

$$\begin{aligned}
\iff\;& \langle u,c\rangle \in \bowtie_p(R_1,...,R_i,...,R_n) \wedge \langle u,c\rangle \notin \bowtie_p(R_1,...,\triangledown R_i,...,R_n) \\
\iff\;& \exists r \in \bowtie_p(R_1,...,R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T} \wedge r.\mathtt{T} \Leftrightarrow h(\bowtie_p(R_1,...,\triangledown R_i,...,R_n), r, A_R) = r.\mathtt{T} \\
\iff\;& \exists r \in \bowtie_p(R_1,...,R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T} \Leftrightarrow h(\bowtie_p(R_1,...,\triangledown R_i,...,R_n), r, A_R) \neq \varnothing \\
\iff\;& \langle u,c\rangle \in \bowtie_p(R_1,...,R_i,...,R_n) \Leftrightarrow \bowtie_p(R_1,...,\triangledown R_i,...,R_n) \\
\iff\;& \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,R_i,...,R_n) \Leftrightarrow \bowtie_p(R_1,...,\triangledown R_i,...,R_n)).
\end{aligned}$$

To prove (c.16), it suffices to show that $\theta(\bowtie_p(R_1,...,R_i \cup \triangle R_i,...,R_n)) =_{nt} \theta(\bowtie_p(R_1,...,R_i,...,R_n) \cup \bowtie_p(R_1,...,\triangle R_i,...,R_n))$:

$$\begin{aligned}
& \theta(\bowtie_p(R_1,...,R_i \cup \triangle R_i,...,R_n)) \\
=_{nt}\;& \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1),...,\theta(R_i \cup \triangle R_i),...,\theta(R_n))) \\
=_{nt}\;& \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1),...,\theta(R_i) \cup^{nt} \theta(\triangle R_i),...,\theta(R_n))) \\
=_{nt}\;& \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1),...,\theta(R_i),...,\theta(R_n)) \cup^{nt} \bowtie^{nt}_{p'}(\theta(R_1),...,\theta(\triangle R_i),...,\theta(R_n))) \\
=_{nt}\;& \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1),...,\theta(R_i),...,\theta(R_n))) \cup^{nt} \pi^{nt}_{A'}(\bowtie^{nt}_{p'}(\theta(R_1),...,\theta(\triangle R_i),...,\theta(R_n))) \\
=_{nt}\;& \theta(\bowtie_p(R_1,...,R_i,...,R_n)) \cup^{nt} \theta(\bowtie_p(R_1,...,\triangle R_i,...,R_n)).
\end{aligned}$$

It remains to be proven that:

$$\begin{aligned}
& \theta(\bowtie_p(R_1,...,R_i,...,R_n)) \cup^{nt} \theta(\bowtie_p(R_1,...,\triangle R_i,...,R_n)) \\
=_{nt}\;& \theta(\bowtie_p(R_1,...,R_i,...,R_n) \cup \bowtie_p(R_1,...,\triangle R_i,...,R_n)).
\end{aligned}$$

To this end, we shall first prove a useful fact:

$$\begin{aligned}
& \theta(R_i \cup \triangle R_i) =_{nt} \theta(R_i) \cup^{nt} \theta(\triangle R_i) \\
\Longrightarrow\;& (\forall r \in \bowtie_p(R_1,...,R_i,...,R_n) : r.\mathtt{T} \cup h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R) = r.\mathtt{T}) \\
& \wedge (\forall r \in \bowtie_p(R_1,...,\triangle R_i,...,R_n) : r.\mathtt{T} \cup h(\bowtie_p(R_1,...,R_i,...,R_n), r, R_A) = r.\mathtt{T}).
\end{aligned}$$

Because of symmetry, we only need to show that:

$$\forall r \in \bowtie_p(R_1,...,R_i,...,R_n) : r.\mathtt{T} \cup h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R) = r.\mathtt{T}.$$

If $h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R) = \varnothing$, the fact obviously holds. Otherwise, let r be the result of temporally joining together $r_1 \in R_1,...,r_i \in R_i,...,r_n \in R_n$. Then $\exists r'_i \in \triangle R_i : r'_i.A_{R_i} = r_i.A_R \wedge h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R) = r_1.\mathtt{T} \cap ... \cap r'_i.\mathtt{T} \cap ... \cap r_n.\mathtt{T}$. By Lemma A.2, $r_i.\mathtt{T} \cup r'_i.\mathtt{T} = r_i.\mathtt{T}$. Therefore, $r.\mathtt{T} \cup h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R) = (r_1.\mathtt{T} \cap ... \cap r_i.\mathtt{T} \cap ... \cap r_n.\mathtt{T}) \cup (r_1.\mathtt{T} \cap ... \cap r'_i.\mathtt{T} \cap ... \cap r_n.\mathtt{T}) = r_1.\mathtt{T} \cap ... \cap (r_i.\mathtt{T} \cup r'_i.\mathtt{T}) \cap ... \cap r_n.\mathtt{T} = r_1.\mathtt{T} \cap ... \cap r_i.\mathtt{T} \cap ... \cap r_n.\mathtt{T} = r.\mathtt{T}$. Using this result, we can now complete the proof of (c.16):

$$\begin{aligned}
& \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,R_i,...,R_n)) \cup^{nt} \theta(\bowtie_p(R_1,...,\triangle R_i,...,R_n)) \\
\iff\;& \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,R_i,...,R_n)) \vee \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,\triangle R_i,...,R_n)) \\
\iff\;& \langle u,c\rangle \in \bowtie_p(R_1,...,R_i,...,R_n) \vee \langle u,c\rangle \in \bowtie_p(R_1,...,\triangle R_i,...,R_n) \\
\iff\;& (\exists r \in \bowtie_p(R_1,...,R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T}) \\
& \vee (\exists r \in \bowtie_p(R_1,...,\triangle R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T}) \\
\iff\;& (\exists r \in \bowtie_p(R_1,...,R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T} \cup h(\bowtie_p(R_1,...,\triangle R_i,...,R_n), r, A_R)) \\
& \vee (\exists r \in \bowtie_p(R_1,...,\triangle R_i,...,R_n) : u = r.A_R \wedge c = r.\mathtt{T} \cup h(\bowtie_p(R_1,...,R_i,...,R_n), r, A_R)) \\
\iff\;& \langle u,c\rangle \in \bowtie_p(R_1,...,R_i,...,R_n) \cup \bowtie_p(R_1,...,\triangle R_i,...,R_n) \\
\iff\;& \langle u,c\rangle \in \theta(\bowtie_p(R_1,...,R_i,...,R_n) \cup \bowtie_p(R_1,...,\triangle R_i,...,R_n)).
\end{aligned}$$

□

Proof of Theorem 4.5. Follows from Theorems 3.1, 3.3 and Definition 3.2. By Theorem 3.1, it suffices to show that $\forall t', t_0 \leq t' \leq t_{now} : \tau_{t'}(op(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t))) =_{nt} \tau_{t'}(\psi_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t))$. For $t_0 \leq t' \leq t$:

$$\begin{aligned}
& \tau_{t'}(op(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t))) \\
=_{nt} \;& op^{nt}(\tau_{t'}(\psi_t([\mathcal{E}_1]_t)), ..., \tau_{t'}(\psi_t([\mathcal{E}_n]_t))) \\
=_{nt} \;& op^{nt}(\tau_{t'}([\mathcal{E}_1]_t), ..., \tau_{t'}([\mathcal{E}_n]_t)) \\
=_{nt} \;& [op^{nt}(\tau_{t'}([\mathcal{E}_1]_t), ..., \tau_{t'}([\mathcal{E}_n]_t))]_t \\
=_{nt} \;& [op^{nt}(\tau_{t'}(\mathcal{E}_1), ...\tau_{t'}(\mathcal{E}_n))]_t \\
=_{nt} \;& [\tau_{t'}(op(\mathcal{E}_1, ..., \mathcal{E}_n))]_t \\
=_{nt} \;& \tau_{t'}([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \\
=_{nt} \;& \tau_{t'}(\psi_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t)).
\end{aligned}$$

For $t < t' \leq t_{now}$:

$$\begin{aligned}
& \tau_{t'}(op(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t))) \\
=_{nt} \;& op^{nt}(\tau_{t'}(\psi_t([\mathcal{E}_1]_t)), ..., \tau_{t'}(\psi_t([\mathcal{E}_n]_t))) \\
=_{nt} \;& op^{nt}(\tau_t([\mathcal{E}_1]_t), ..., \tau_t([\mathcal{E}_n]_t)) \\
=_{nt} \;& [op^{nt}(\tau_t([\mathcal{E}_1]_t), ..., \tau_t([\mathcal{E}_n]_t))]_t \\
=_{nt} \;& [op^{nt}(\tau_t(\mathcal{E}_1), ...\tau_t(\mathcal{E}_n))]_t \\
=_{nt} \;& [\tau_t(op(\mathcal{E}_1, ..., \mathcal{E}_n))]_t \\
=_{nt} \;& \tau_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \\
=_{nt} \;& \tau_{t'}(\psi_t([op(\mathcal{E}_1, ..., \mathcal{E}_n)]_t)).
\end{aligned}$$

□

Proof of Theorem 4.6. Follows from Equality (c.14) given in Theorem 4.4, and the definitions of σ_p and $\bowtie_p$. To prove (r.6), we begin with the right-hand side:

$$\begin{aligned}
& (\psi_t([\sigma_p(\mathcal{E})]_t) \Leftrightarrow \sigma_{\neg p}(\psi_t([\sigma_p(\mathcal{E})]_t))) \cup \sigma_p(\psi_t([\sigma_{\neg p}(\mathcal{E})]_t)) \\
= \;& \sigma_p(\psi_t([\sigma_p(\mathcal{E})]_t)) \cup \sigma_p(\psi_t([\sigma_{\neg p}(\mathcal{E})]_t)).
\end{aligned}$$

It is easy to verify that $\theta(\psi_t([\sigma_p(\mathcal{E})]_t) \cup \psi_t([\sigma_{\neg p}(\mathcal{E})]_t)) =_{nt} \theta(\psi_t([\sigma_p(\mathcal{E})]_t)) \cup^{nt} \theta(\psi_t([\sigma_{\neg p}(\mathcal{E})]_t))$. By (c.14):

$$\begin{aligned}
& \sigma_p(\psi_t([\sigma_p(\mathcal{E})]_t)) \cup \sigma_p(\psi_t([\sigma_{\neg p}(\mathcal{E})]_t)) \\
= \;& \sigma_p(\psi_t([\sigma_p(\mathcal{E})]_t) \cup \psi_t([\sigma_{\neg p}(\mathcal{E})]_t)) \\
= \;& \sigma_p(\psi_t([\mathcal{E}]_t)).
\end{aligned}$$

To prove (r.7), let:

$$\begin{aligned}
\mathcal{R} &= \bowtie_p(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)); \\
\mathcal{R}' &= \psi_t([\bowtie_p(\mathcal{E}_1, ..., \mathcal{E}_n)]_t); \\
\nabla\mathcal{R} &= \bowtie_{\neg p}(\psi_t([\rhd\!\!<_p(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n)]_t), \psi_t([\rhd\!\!<_p(\mathcal{E}_2, \mathcal{E}_1, ..., \mathcal{E}_n)]_t), ..., \psi_t([\rhd\!\!<_p(\mathcal{E}_n, \mathcal{E}_1, ..., \mathcal{E}_{n-1})]_t)); \\
\triangle\mathcal{R} &= \bowtie_p(\psi_t([\rhd\!\!<_{\neg p}(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_n)]_t), \psi_t([\rhd\!\!<_{\neg p}(\mathcal{E}_2, \mathcal{E}_1, ..., \mathcal{E}_n)]_t), ..., \psi_t([\rhd\!\!<_{\neg p}(\mathcal{E}_n, \mathcal{E}_1, ..., \mathcal{E}_{n-1})]_t)).
\end{aligned}$$

It suffices to show that: (a) $r \in \mathcal{R} \Longrightarrow r \notin \bigtriangledown\mathcal{R}$; (b) $r \in \mathcal{R}' \wedge r \notin \mathcal{R} \Longrightarrow r \in \bigtriangledown\mathcal{R}$; (c) $r \notin \mathcal{R} \Longrightarrow r \notin \bigtriangleup\mathcal{R}$; (d) $r \notin \mathcal{R}' \wedge r \in \mathcal{R} \Longrightarrow r \in \bigtriangleup\mathcal{R}$. (a) and (c) are obvious. We will first prove (b):

$$\begin{aligned}
& r \in \mathcal{R}' \wedge r \notin \mathcal{R} \\
\Longrightarrow\quad & r \in \psi_t([\bowtie_{true}(\rtimes_p(\mathcal{E}_1, ..., \mathcal{E}_n), ..., \rtimes_p(\mathcal{E}_n, ..., \mathcal{E}_{n-1}))]_t) \wedge r \notin \mathcal{R} \\
\Longrightarrow\quad & r \in \bowtie_{true}(\psi_t([\rtimes_p(\mathcal{E}_1, ..., \mathcal{E}_n)]_t), ..., \psi_t([\rtimes_p(\mathcal{E}_n, ..., \mathcal{E}_{n-1})]_t)) \\
& \wedge r \notin \bowtie_p(\psi_t([\rtimes_p(\mathcal{E}_1, ..., \mathcal{E}_n)]_t), ..., \psi_t([\rtimes_p(\mathcal{E}_n, ..., \mathcal{E}_{n-1})]_t)) \\
\Longrightarrow\quad & r \in \bigtriangledown\mathcal{R}.
\end{aligned}$$

We now prove (d):

$$\begin{aligned}
& r \notin \mathcal{R}' \wedge r \in \mathcal{R} \\
\Longrightarrow\quad & r \notin \mathcal{R}' \wedge r \in \mathcal{R} \wedge r \in \bowtie_{true}(\psi_t([\mathcal{E}_1]_t), ..., \psi_t([\mathcal{E}_n]_t)) \\
\Longrightarrow\quad & r \notin \mathcal{R}' \wedge r \in \mathcal{R} \wedge r \in \psi_t([\bowtie_{true}(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \\
\Longrightarrow\quad & r \in \mathcal{R} \wedge r \in \psi_t([\bowtie_{\neg p}(\mathcal{E}_1, ..., \mathcal{E}_n)]_t) \\
\Longrightarrow\quad & r \in \mathcal{R} \wedge r \in \psi_t([\bowtie_{true}(\rtimes_{\neg p}(\mathcal{E}_1, ..., \mathcal{E}_n), ..., \rtimes_{\neg p}(\mathcal{E}_n, ..., \mathcal{E}_{n-1}))]_t) \\
\Longrightarrow\quad & r \in \mathcal{R} \wedge r \in \bowtie_{true}(\psi_t([\rtimes_{\neg p}(\mathcal{E}_1, ..., \mathcal{E}_n)]_t), ..., \psi_t([\rtimes_{\neg p}(\mathcal{E}_n, ..., \mathcal{E}_{n-1})]_t)) \\
\Longrightarrow\quad & r \in \bigtriangleup\mathcal{R}.
\end{aligned}$$

□