Analysing Temporal Relations – Beyond Windows, Frames and Predicates

Simon Brodt
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
simon.brodt@ifi.lmu.de

François Bry
Institute for Informatics
University of Munich
http://www.pms.ifi.lmu.de/
bry@lmu.de

ABSTRACT

This article proposes an approach to rely on the standard operators of relational algebra (including grouping and aggregation) for processing complex event without requiring window specifications. In this way the approach can process complex event queries of the kind encountered in applications such as emergency management in metro networks. This article presents Temporal Stream Algebra (TSA) which combines the operators of relational algebra with an analysis of temporal relations at compile time. This analysis determines which relational algebra queries can be evaluated against data streams, i. e. the analysis is able to distinguish valid from invalid stream queries. Furthermore the analysis derives functions similar to the pass, propagation and keep invariants in Tucker's et al. "Exploiting Punctuation Semantics in Continuous Data Streams". These functions enable the incremental evaluation of TSA queries, the propagation of punctuations, and garbage collection. The evaluation of TSA queries combines bulk-wise and out-of-order processing which makes it tolerant to workload bursts as they typically occur in emergency management. The approach has been conceived for efficiently processing complex event queries on top of a relational database system. It has been deployed and tested on MonetDB.

Categories and Subject Descriptors

F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*Algebraic approaches to semantics, Operational semantics, Program analysis*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms

Theory, Algorithms, Languages, Verification

Keywords

CEP, Event Processing, DSMS, Relational Algebra, Temporal Analysis, Punctuations, Frames, Predicate Windows

1. INTRODUCTION

Data stream management systems (DSMS) widely use tumbling, sliding, landmark and similar kinds of windows [4, 9, 20, 21] for handling "blocking operators" like negation or aggregation and "unbounded stateful operators" like join [25]. All these windows have in common, that they either have fixed bounds (tumbling/sliding windows) or last from a fixed bound to the present (landmark windows) and therefore are data independent. However a number of new applications, among others temperature, traffic and network monitoring [16, 23] require a data dependent way of detecting the "episodes of interest" [23] or "windows-of-interest" [16] within a data stream.

In emergency management applications like in metro networks, at airports or in power grids [30, 31] the relevant "episode" or part of a data stream is frequently defined by the start and the end of an emergency like a fire. Of course those queries responsible for detecting the start of an emergency need to be evaluated constantly and thus all or at least most of the data stream is relevant for those rules. However analytic queries like "the maximum smoke concentration within a room during a fire" should only be evaluated when actually needed i. e. in the presence of the corresponding emergency. Both the starting time as well as the temporal extension of the analysis is determined by events (the start and end of an emergency) and cannot[1] be defined in advance. Moreover, the precise knowledge of the start and end of some analysis period is sometimes interesting in itself, e. g. when monitoring an evacuation. Thus data independent window definitions are hardly suited for this kind of queries.

"Frames" [23] and "predicate-windows" [16] are two approaches towards detecting relevant parts of a data stream in an adaptive, data-dependent way. Interestingly both approaches introduce new operators although the standard operators of relational algebra[2] [2, 15] are actually able to express queries like the ones sketched for emergency management applications and (apparently) also those described in [23] and [16]. Instead of new operators we propose using the original operators of relational algebra for data stream processing. The challenge is that some relational algebra queries cannot be evaluated on data streams,[3] i. e., are invalid. We address this challenge by proposing an in-depth compile-time analysis of temporal relations which first can distinguish valid from invalid stream queries and second en-

[1] Unfortunately not. Knowing the time and extent of an emergency in advance would be great.
[2] including grouping and aggregation
[3] Or are not even well-defined like e. g. an infinite sum.

ables an incremental evaluation of relational algebra expressions on data streams.

The contributions of this article are as follows:

- We introduce a common data model for data streams and static relations generalizing the data model of relational algebra.
- We define Temporal Stream Algebra (TSA) which enhances the standard operators[2] of relational algebra by an mechanism for propagating constraints on temporal relations between attributes and on the correlation of attributes to the stream progress within the stream schema.
- We present an analysis for the propagated constraints that can determine the validity of TSA queries and derives functions similar to the invariants of [25] that define first when results can be passed on, second how information on stream progress is propagated and third which data needs to be kept.
- Based on these functions we finally describe an bulk-wise and out-of-order evaluation for TSA queries reducing the evaluation of TSA queries on data streams to a repeated evaluation of ordinary relational algebra expressions.

The rest of this article is organized as follows: Section 2 motivates our approach based on examples from emergency management. Section 3 presents our data model for data streams and introduces a schema for data streams based on constrains on temporal relations between attributes and on the correlation of attributes to stream progress. Section 4 describes the propagation of the constraints through the operators of relational algebra and gives a definition of formally "valid" queries. Section 4 also shows that "valid" queries can in fact be evaluated on data streams. Section 5 illustrates how different kinds of queries can be expressed in TSA. Section 6 describes the incremental evaluation for TSA queries. Section 7 discusses relevant prior work and Section 8 points to the presented achievements and future work.

2. EMERGENCY MANAGEMENT

Emergency management applications typically consist of three major kinds of rules or queries: First *detection rules* which detect the start and the end[4] of an emergency. Second *analytic rules* which in case of an emergency gather and analyse further information on the emergency as to provide a comprehensive assessment of the emergency situation. Finally *reactive rules* are used to execute or propose[5] proper reactions to an emergency [29, 30, 31]. Within this article we focus on the interplay of detection and analytic rules.

Event-controlled aggregation. Detection queries, like the detection of fire, need to be evaluated constantly. By contrast analytic queries only become important in the presence of the corresponding emergency. For example, there is hardly any use in computing the maximum smoke concentration in a room in absence of a fire. However in case of a fire, knowing the maximum smoke concentration of a room is very important. Smoke is actually the greatest treat for the life of people[29]. With the beginning of a fire it is therefore required to constantly evaluate the maximum smoke concentration of a room as to choose and adapt safe, i. e. smoke free, evacuation paths and an appropriate ventilation regime. When the fire-fighters arrive they (manually) take over both the further evacuation and the ventilation regime in particular because they bring along a lot of own equipment that is not connected to the emergency management system.[6] Thus the constant evaluation of the maximum smoke concentration becomes irrelevant. However the fire-fighter might be interested in a query like "the maximum smoke concentration within a room from the beginning of the fire to the arrival of the fire-brigade" giving a first impression of the quality of the so far evacuation management.

Obviously neither the beginning of a fire nor the precise time until the arrival of the fire-brigade are known in advance. Thus data-independent windows, e. g. tumbling or sliding windows, hardly seem to be suited for this kind of scenario. Instead the aggregation, i. e. the computation of the maximum smoke concentration, needs to be controlled by events indicating the start and the end of the aggregation, i. e. the detection of a fire and the arrival of the fire-brigade.

Access to Static Relations. Both detection and analytic queries frequently need to access static or hardly changing relations for correctly interpreting the incoming events. For example sensor messages usually carry sensor ids, but do not necessarily carry data indicating the locations of issuing sensors. The location is however essential for correlating messages from different sensors. So there should be an easy way to attach the location to the incoming sensor messages. More complex topological information is needed for handling an emergency in a power-grid but also in case of a fire in metro stations and airports[31] or other buildings. For example neighbouring rooms of a burning area, are directly threatened by the fire. Identifying those rooms needs to correlate the event of fire detection with the static neighbour relation of rooms.

User-defined Timestamps for Composite Events. In emergency management the "right" choice for the timestamp of an composite event is not always obvious and may differ from query to query. Consider for example the following two rules: Rule 1: A temperature sensor is considered to be malfunctioning if there is an message from that sensor which is not followed by another message within 30 seconds. Rule 2: A precautionary fire alarm is raised if there is a potential fire detection in an area and no report from the responsible warden is received within 2 minutes after the potential fire detection. Both queries have the same basic structure, a positive event that triggers the rule if it is not followed by another event within a certain time-frame. However for the first query, the timestamp for the "malfunction sensor" event should probably be the time of the first missing messages, e. g. 10 seconds after the last message from that sensor. By contrast the time for the precautionary alarm should be the time of the (potential) fire detection and not the time of the missing report. Therefore the user should be able to choose the most meaningful definition for the timestamps of a composite events on a query per query basis .

Multiple Time Lines. Emergency management requires queries referring to timestamps according to multiple different timelines. In a simple case these timelines are just application *and* system time. Reconsider the rule – A precautionary fire alarm is raised if there is a potential fire detec-

[4] Usually a human operator needs to confirm the end of an emergency. However the system might propose that an emergency could be marked as being finished.

[5] Some actions will need operator approval.

[6] Actually we learned that the fire-fighters will not entrust their lives to any foreign emergency management system but prefer to do their own reconnaissance.

tion in an area and no report from the responsible warden is received within 2 minutes after the potential fire detection. The timespan of 2 minutes is defined between the detection of the potential fire, i. e. application time, and the reception of the report, i. e. system time. As to ensure that no more than two minutes will elapse between the potential detection and a reaction to that detection it is crucial that the 2 minutes period is defined in exactly that way, as transmission delays could otherwise block the emergency management system.

Further timelines are introduced by simulations used for predicting the physical evolution of an emergency, e. g. smoke propagation, which is very important for taking the right decisions, e. g. choosing safe evacuation routes. Beside application and system time, events returned by such simulations carry a timestamp denoting the time for which the prediction is meant to hold. Multiple timelines are also useful for first implementing slack [1] on a query per query bases and and in a way transparent for the user and second for skipping the derivation of composite events which have become irrelevant by lapse of time e. g. due to prioritization in favor of more important queries.

Supporting multiple timelines requires more than having multiple timestamp attributes as different timelines impose different orders on the events. Therefore in-order processing is only possible for at most one of the timelines. For all other timelines the evaluation is inevitably out-of-order. Thus, beside other benefits, out-of-order processing [21], seems to be almost mandatory for supporting multiple timelines.

Relational Algebra. The operators of relation algebra[2] turned out be sufficient for expressing queries like the ones described above.[7] Thus it seemed natural to generalize the operators of relational algebra towards data streams. Using the operators of relational algebra has a number of advantages: The operators form a good basis for the implementation of a high-level declarative language like for example the event, state and action language Dura [28, 26, 27]. Data streams can be queried in the same way that is already familiar from database relations. The properties of the operators are well understood. For example the laws on operator permutations are very important for the optimization of relational algebra expressions. There exist reliable techniques for the evaluation and optimization of relational algebra expressions, like specialized join algorithms, heuristics for operator reordering or cost-models, that base on these laws and on other properties of the operators of relational algebra. Preserving the operators and their characteristics means that these techniques and algorithms can be reused or easily adopted for data streams.

3. STREAMS & TEMPORAL RELATIONS

A key observations on data streams made in [21] is that, though they might arrive out-of-order with respect to any of their attributes, data streams usually make "progress" with respect to some of their attributes. More precisely a a data stream is making "progress" on an attribute a if it eventually exceeds any value v for that attribute. This lead to the definition of so-called "progressing streams" [21]. However the original definition of *progressing streams* is not fully suited for our purposes. First the cross-product of two progressing streams is not a progressing stream itself, second static relations are not covered, and third the definition implies a one-by-one arrival of tuples inducing some strict (though arbitrary) order on the tuples.[8]

Thus, we use the more general notion of *temporal streams*. In fact all progressing streams are temporal streams and a progressing stream its corresponding temporal stream have the same progressing attributes.[9] Temporal streams differ from progressing streams as follows:

1. Prefixes of a data stream are defined with respect to time and not to tuple count
2. Data streams may be static, i. e. all tuples are available from the very beginning of the data stream
3. Progressing attributes are generalized to *progressing sets of attributes*

DEFINITION 1. **(Temporal Streams)**

1. A *data stream* R with attribute schema $\mathscr{A}(R) \subseteq ATTR$ ($ATTR$ is a set of attribute names) is a (possibly infinite) relation with finite prefix R^p at each point in time $p \in \mathbb{Q}$, i. e.
$$R = \bigcup_{p \in \mathbb{Q}} R^p \text{ with } p_1 \leq p_2 \Rightarrow R^{p_1} \subseteq R^{p_2} \text{ and } |R^p| < +\infty \text{ for all } p, p_1, p_2 \in \mathbb{Q}$$
2. R is *static* iff $R^p = R$ for all $p \in \mathbb{Q}$
3. $\emptyset$ is a *progressing set of attributes for R* iff R is static
4. $\{a_1, \ldots, a_k\} \neq \emptyset$ is a *progressing set of attributes for R* iff $\{a_1, \ldots, a_k\} \subseteq \mathscr{A}(R)$ and R eventually exceeds any upper bound $s_1, \ldots, s_k \in \mathbb{Q}$ for $a_1, \ldots, a_k$, i. e.
$\exists p \in \mathbb{Q} : \{r \in R \mid r(a_1) \leq s_1 \wedge \cdots \wedge r(a_k) \leq s_k\} \subseteq R^p$
5. A data stream R is a *temporal stream* iff R has a progressing set of attributes

Progressing sets of attributes play quite a hidden role in the rest of the article although they are absolutely required. For input and output relations we are mostly interested in progressing sets of attributes with a single element, i. e. the so-called progressing attributes.[9] Progressing sets of attributes are important particularly for intermediate results like the cross-product of two temporal streams and for the propagation of information on progressing attributes through the query expression.

The approach is centered around the following observation: If the progressing attributes of the temporal input streams of a relational algebra expression and the temporal relations specified in the expression imply at least one progressing attribute for the output stream then the expression is valid and can be incrementally evaluated on temporal streams. In fact there exists a (monotone) function that computes the minimum achievable progress of the output stream with respect to its progressing attributes based on the progress of the input streams with respect to their progressing attributes.

In the rest of this section we introduce the ingredients of a schema for temporal streams that can be used to describe, propagate and analyse information on progressing attributes and temporal relations.

[7] This does not mean that using the operators of relational algebra is a requirement for emergency management. It seems to be good choice, though.

[8] Assuming a one-by-one arrival of events or tuples seems to be artificial in parallel or even distributed environments and the implicit postulation of an order (even an arbitrary one) seems to be inconvenient for a model of unordered streams.

[9] An attribute a is called a *progressing attribute* of a temporal stream iff $\{a\}$ is a progressing set of attribute.

3.1 Temporal Relations

Temporal relations play a major role with respect to progressing attributes[9] (or progressing sets of attributes). For example if an attribute a_1 is a progressing attribute for a temporal stream R and attribute a_2 is known to be greater than a_1 for all tuples of R, e. g. after a selection $\sigma[a_1 \leq a_2]$, then a_2 is also a progressing attribute for R. Even more if neither a_1 nor a_2 are progressing attributes for R, but $\{a_1, a_2\}$ is a progressing set of attributes for R, then a_2 is a progressing attribute for the derived stream $\sigma[a_1 \leq a_2](R)$.

Temporal relations are also introduced when new relative timestamps, e. g. for a composite event, are defined from existing ones. Consider the following definition of a new timestamp $a = \max(a_1, a_2)$. In that case it is know that a is greater than both a_1 and a_2 and therefore if $\{a_1, a_2\}$ is a progressing set of attributes for R then a is progressing attribute for $\iota[a = \max(a_1, a_2)](R)$ the stream with the newly introduced timestamp a.

In TSA new relative timestamps and explicit temporal relations are are specified by means of *temporal terms* and *temporal relations formulas* (TRFs). Temporal terms can shift timestamps by a constant amount of time, they can express the maximum or minimum of a number of timestamps and they can derive less precise versions of a timestamp, e. g. $floor_{1min}(t)$ yields the last preceding full minute for timestamp t. The latter can be used to specify tumbling windows in a very elegant way (Section 5). Furthermore temporal terms can be nested. Beside attributes temporal terms may contain variables which are used when operators like projection discard attributes. In that case variables help to efficiently preserves transitive temporal relations. For example if $a_1 \leq a_2 \leq a_3$ and a_2 is discarded by a projection then $a_1 \leq v_{a_2} \leq a_2$ preserves the transitive relation $a_1 \leq a_2$ between a_1 and a_3.[10]

Definition 2. **(Temporal Terms)**
Temporal terms are defined inductively:
1. $v \in ATTR \cup VAR$ is an atomic temporal term
2. $t + c$ is a temporal term if t is a temporal term and $c \in \mathbb{Q}$
3. $min(t_1, \ldots, t_k)$, $max(t_1, \ldots, t_k)$ are temporal terms if $t_1, \ldots, t_k$ are temporal terms
4. $floor_c(t)$, $ceil_c(t)$ are temporal terms if t is a temporal term and $c \in \mathbb{Q}$

where VAR is a set of variables and $VAR \cap ATTR = \emptyset$.

TRFs consist of comparisons of timestamps defined by temporal terms which can be combined by conjunction, disjunction and negation. During the propagation of TRFs through the operators of TSA we sometimes need to express that a variable is not part of the namespace of a particular part of a formula. This is done by atoms $ignore(v)$.[11] .

[10] Making the transitive relations explicit can cause a significant blowup of the formula.

[11] A full explanation on that point is out of the scope of this article. The basic reason for this is that the "variables" of temporal terms are used with two different semantics with respect to first order predicate logic, namely first as existentially quantified variables when the relations between attributes are concerned (compare Definition 9) or second as logic constants when the temporal distance (Definition 5) between *attributes and "variables"* is required. It is only when "variables" are used in the sense of logic constants that the formulas $ignore(v)$ have a meaning.

Definition 3. **(Temporal Relation Formulas)**
Temporal relation formulas (TRFs) are defined inductively:
1. $\top$ and $\bot$ are atomic TRFs
2. $t_1 \,\mathsf{op}\, t_2$ is an atomic TRF for $\mathsf{op} \in \{<, \leq, =, \geq, >, \neq\}$ if t_1 and t_2 are temporal terms
3. $ignore(v)$ is an atomic TRF if $v \in VAR$ is a variable
4. $G \wedge G'$, $G \vee G'$ and $\neg G$ are TRFs if G, G' are TRFs

Despite of the complex specifications that are possible using general TRFs they can be normalized to TRFs with a comparably simple structure. This is essential for the algorithmic analysis of the temporal relations. Note that the size of the normalized formula G^{norm} is only linear in the size[12] of the original formula G.

Definition 4. **(Normalized TRFs)** A TRF G can be normalized to a TRF G^{norm} using the following equivalences and implications in left to right direction. The normalized TRF G^{norm} is equal to $\top$ or $\bot$, or all atomic subformulas of G^{norm} have the form $v \leq w + c$ or $ignore(v)$ and negation occurs at most at literals of the form $\neg ignore(v)$.[13]

$$
\begin{aligned}
Neg1 &: \neg(G \wedge G') \Leftrightarrow (\neg G \vee \neg G') \\
Neg2 &: \neg(G \vee G') \Leftrightarrow (\neg G \wedge \neg G') \\
Neg3 &: \neg\neg G \Leftrightarrow G \\
Neg4 &: \neg(t_1 \,\mathsf{op}\, t_2 + c) \;\Leftrightarrow\; t_1 \,\mathsf{op}^{-1}\, t_2 + c \\
Neg5 &: \neg\top \Leftrightarrow \bot \text{ and } \neg\bot \Leftrightarrow \top \\
Top &: G \wedge \top \Leftrightarrow G \text{ and } G \vee \top \Leftrightarrow \top \\
Bot &: G \wedge \bot \Leftrightarrow \bot \text{ and } G \vee \bot \Leftrightarrow G \\
Zero &: v \,\mathsf{op}\, w \;\Leftrightarrow\; v \,\mathsf{op}\, w + 0 \\
Eq &: t_1 = t_2 + c \;\Leftrightarrow\; (t_1 \leq t_2 + c) \wedge (t_1 \geq t_2 + c) \\
Neq &: t_1 \neq t_2 + c \;\Leftrightarrow\; (t_1 < t_2 + c) \vee (t_1 > t_2 + c) \\
Geq &: t_1 \geq t_2 + c \;\Leftrightarrow\; t_2 + c \leq t_1 \\
Gr &: t_1 > t_2 + c \;\Leftrightarrow\; t_2 + c < t_1 \\
Less^{14} &: t_1 < t_2 + c \;\Leftrightarrow\; t_1 \leq t_2 + (c - \varepsilon) \\
Arith1 &: t_1 + c \,\mathsf{op}\, t_2 \;\Leftrightarrow\; t_1 \,\mathsf{op}\, t_2 + (-c) \\
Arith2 &: t_1 \,\mathsf{op}\, (t_2 + c) + d \;\Leftrightarrow\; t_1 \,\mathsf{op}\, t_2 + (c + d) \\
Min1 &: t \leq min\{t_1, \ldots, t_k\} + c \;\Leftrightarrow \\
&\quad t \leq t_1 + c \wedge \ldots \wedge t \leq t_k + c \\
Min2 &: min(t_1, \ldots, t_k) \leq t + c \;\Leftrightarrow \\
&\quad t_1 \leq t + c \vee \ldots \vee t_k \leq t + c \\
Max1 &: t \leq max(t_1, \ldots, t_k) + c \;\Leftrightarrow \\
&\quad t \leq t_1 + c \vee \ldots \vee t \leq t_k + c \\
Max2 &: max(t_1, \ldots, t_k) \leq t + c \;\Leftrightarrow \\
&\quad t_1 \leq t + c \wedge \ldots \wedge t_k \leq t + c \\
Floor1 &: floor_c(t_1) \leq t_2 \;\Rightarrow\; t_1 - c < t_2 \\
Floor2 &: t_2 \leq floor_c(t_1) \;\Rightarrow\; t_2 \leq t_1 \\
Ceil1 &: ceil_c(t_1) \leq t_2 \;\Rightarrow\; t_1 \leq t_2 \\
Ceil2 &: t_2 \leq ceil_c(t_1) \;\Rightarrow\; t_2 < t_1 + c
\end{aligned}
$$

for temporal terms $t_1, \ldots, t_k$ and $c, d \in \mathbb{Q}$ and TRFs G, G' and $v, w \in ATTR \cup VAR$ and $\mathsf{op} \in \{<, \leq, =, >=, >, \neq\}$ and $<^{-1} \mapsto >$, $\leq^{-1} \mapsto \geq$, $=^{-1} \mapsto \neq$, $\geq^{-1} \mapsto \leq$, $>^{-1} \mapsto <$, $\neq^{-1} \mapsto =$

A basic algorithm for the normalization of a TRF is very simple: Whenever the algorithm finds a syntactical match for the left side of one of the above equivalences and implications, then the matching subformula is replaced by the right side of the equivalence or implication. The algorithm stops if it does not find any further matches for the left side

[12] Size in number of terms not in number of atoms.

[13] Literals of the form $\neg ignore(v)$ do not occur in practice.

[14] Think of ε as an infinitely small value such that $c - \varepsilon < c$ but $d < c - \varepsilon$ if $d < c$ and $(c - \varepsilon) + (d - \varepsilon) = ((c + d) - \varepsilon)$.

of any of the equivalences or implications. The algorithm will always terminate with normalized form of the TRF. It is most efficient if it uses the equivalences and implications in top down precedence.

The temporal distance[15] of two variables or attributes w and v with respect to a TRF is the maximum value that the difference $v - w$ can take, i. e.$v - w \leq dist(w, v)$ considering the restriction imposed by the TRF. The temporal distance is a core element of the compile-time analysis of temporal relations and is used to construct the functions (Definition 25) that propagate information on the progress of the inputs streams of a query to the information on the progress of the output stream of a query. Temporal distances can also be used for garbage collection (see end of Section 6) for which they have been proposed first [7]. The following definition significantly generalizes that of [7].

Definition 5. **(Temporal Distance)** Let G be a temporal relation formula and R be a temporal stream with $\mathscr{A}(R) = A$. The *temporal distance* of two attributes or variables $v, w \in ATTR \cup VAR$ with respect to G is

$$dist_G(w, v) := \max_{C \in dnf(G^{norm})} \{dist_C(w, v)\}$$

$$dist_C(w, v) := \begin{cases} -\infty & \text{if } C \text{ is inconsistent} \\ -\infty & \text{if } C \models ignore(v) \text{ or } C \models ignore(w) \\ \min\{c \mid \mathcal{TD}, C \models_{\overline{\mathbb{Q}}} v \leq w + c\} & \text{else} \end{cases}$$

where $u, v, w \in ATTR \cup VAR$ and $c, d, +\infty, -\infty \in \overline{\mathbb{Q}}$ and G^{norm} is the normalized form of G and C is a conjunction of atomic TRFs of the form $v \leq w + c$ and $\mathcal{TD}$ contains

$$\begin{array}{ll} Ref: & v \leq v \\ Trans: & u \leq v + c \wedge v \leq w + d \Rightarrow u \leq w \pm (c + d) \\ Inf: & v \leq w + +\infty \end{array}$$

The algorithmic analysis of the temporal distances is closely related to the simple temporal problem (STP) [24] and the disjunctive temporal problem DTP [18]. Basically the (naive) analysis algorithm is as follows: The TRF is normalized and converted into disjunctive normal form. Each conjunction is an STP instance. The distance of all pairs of attributes and variables for this instance can determined using any algorithm for the all-pair shortest path problem, e. g. the Floyd Warshall algorithm [14], or specialized algorithms for STP. The distance of two attributes or variables for the whole TRF is then the maximum distance of the two attributes or variables in any of the conjunctions.

3.2 Progressing Sets of Attributes

Stream bound formulas (SBFs) are used to describe the initial progressing sets of attributes of a temporal stream. The stream may have further derived progressing sets of attributes as illustrated in the beginning of Section 3.1. The formulas are called stream *bound* formulas because they actually tell which sets of attributes have the defining property of progressing sets of attributes, i. e. that *upper bounds* for the attributes yield a (finite) prefix of the stream.

Definition 6. **(Stream Bound Formulas)** The set of *stream bound formulas* (SBFs) is defined inductively:

1. $\top$ is an atomic SBF
2. $bounded(v, b)$ is an atomic SBF for $v \in ATTR \cup VAR$ where $b \in BOUND$ is a *stream bound identifier*.
3. $H_1 \wedge H_2$ and $H_1 \vee H_2$ are SBFs iff H_1 and H_2 are SBFs.

[15] Note that the temporal distance is usually asymmetrical.

For example the SBF for a temporal stream R with progressing attributes a_1 and a_2 has the form $bounded(a_1, b_{a_1}) \vee bounded(a_2, b_{a_2})$. The SBF for the cross-product of two temporal stream R and R with progressing attributes a and a' respectively, has the form $bounded(a, b_a) \vee bounded(a', b_{a'})$.

Given a TRF G and a SBF H we can determine all (initial and derived) progressing sets of attributes with respect to G and H in the following way:

Definition 7. **(Progressing Sets of Attributes)** $\{a_1, \ldots, a_k\} \subseteq ATTR$ is a *progressing set of attributes* with respect to a TRF G and a SBF H iff

$$bounded(a_1, b_1), \ldots, bounded(a_k, b_k), G, \mathcal{TD}, \mathcal{SB} \models H$$

for any $b_1, \ldots, b_k \in BOUND$[16] and $\mathcal{SB}$ contains

$$\begin{array}{ll} BD: & v \leq w + c,\ c < +\infty,\ bounded(w, b_w) \Rightarrow bounded(v, b_v) \\ IG: & ignore(v) \Rightarrow bounded(v, b_v) \end{array}$$

for $v, w \in ATTR \cup VAR$, $b_v, b_w \in BOUND$ and $c \in \overline{\mathbb{Q}}$

The algorithmic analysis of the stream bounds is similar to the one for temporal distances. The TRF G is normalized and converted into disjunctive normal form. For each conjunction C the following is done: First the distance between all attributes and variables in the conjunction is computed. Second for each atom in the SBF H, the atom is set to true if the distance from one of the attributes of the potential stream bound $\{p_1, \ldots, p_k\}$ to the attribute or variable in the atom is finite. Otherwise the atom is set to false. If the H holds under this interpretation, then $\{p_1, \ldots, p_k\}$ is a stream bound with respect to C. If $\{p_1, \ldots, p_k\}$ is a stream bound with respect to all conjunctions, then $\{p_1, \ldots, p_k\}$ is a stream bound with respect to G and H.

3.3 Temporal Stream Schema

The schema of a relation in relational algebra is just its set of attributes. Our schema for data streams complements the set of attributes with a TRF describing the temporal relations between the attributes and an SBF identifying the (initial) progressing sets of attributes. Using our stream schema we can propagate this information along the TSA operators and use it determining the validity of a TSA expression and for actually evaluating the expression. So the schema is key element of our approach. Note that our schema for temporal streams covers static relations. Thus we provide common schema for data streams and static relations.

Definition 8. **(Temporal Stream Schema)**

1. A *temporal stream schema* is a triple $S = (A, G, H)$ where $A \subseteq ATTR$ is an attribute schema, G is a TRF with $attr(G) \subseteq A$ and H is a SBF with $attr(H) \subseteq A$
2. S is *static* iff H is equivalent to $\top$
3. $\{a_1, \ldots, a_k\} \subseteq A$ is a *progressing set of attributes* for $S = (A, G, H)$ iff $\{a_1, \ldots, a_k\}$ is a progressing set of attributes with respect to G and H
4. S is *valid* iff S has a progressing set of attributes

The definition of the matching data streams is straight forward. A data streams matches a temporal stream schema if it has the right set of attributes, the temporal relations are as specified in the TRF of the schema and all progressing sets of attributes for the schema are also progressing sets of attributes for the data stream.

[16] Actually the stream bound identifiers of atomic SBFs do not play a role here.

Definition 9. **(Matching Streams)** Let G be a TRF and H be a SBF and let R be a data stream.

1. G holds in R iff for all tuples $r \in R$ the instantiation $\sigma_r(G_{rel})$ of G is satisfiable in $\mathbb{Q}$:
   $\models_{\mathbb{Q}} \exists v_1, \dots, v_l : \sigma_r(G_{rel})$
   where $\mathscr{A}(R) = \{a_1, \dots, a_k\}$ and $\{v_1, \dots, v_l\} = vars(G)$ and $\sigma_r := \{a_1 \mapsto r(a_1), \dots, a_k \mapsto r(a_k)\}$ and G_{rel} results from G when replacing every atom $ignore(v)$ by $\top$ if it occurs with positive polarity or by $\perp$ if it occurs with negative polarity.[17]
2. H holds in R with respect to G, iff all progressing sets of attributes $\{a_1, \dots, a_l\} \subseteq \mathscr{A}(R)$ with respect to G and H are progressing sets of attributes for R.
3. R matches a temporal stream schema $S = (A, G, H)$ iff $\mathscr{A}(R) = A$ and both G and H hold in R.

Proposition 10. If S a *valid* temporal stream schema then every data stream R matching S is a temporal stream.

Proof. As S is valid there is a progressing set of attributes $A_{prog} \subseteq \mathscr{A}(S)$ for S.
R matches S, therefore $\mathscr{A}(R) = \mathscr{A}(S)$ and A_{prog} is a progressing set of attributes for R. $\square$

Temporal stream definitions are used to name the different data streams (usually corresponding to event types) and to provide a formal schema definition for the streams. We impose some restrictions on the schema of temporal stream definitions. Most notably (b) implies that each progressing set of attributes contains at least one progressing attribute. This significantly simplifies the definitions of the functions (Definition 25) propagating information on stream progress from the input streams of a query to its output stream. The restrictions (a) and (c) are of technical nature.

Definition 11. **(Temporal Stream Definition)**
A *temporal stream definition* is a pair $D = (n, S)$ where $n \in STREAM$ is a name for the temporal stream definition, $S = (A, G, H)$ is a valid temporal stream schema and

1. G and H do not contain variables
2. H is a disjunction of atomic SBFs, i. e.
   $H = bounded(a_1, b_1) \vee \ldots \vee bounded(a_k, b_k)$
   for some $a_1, \dots, a_k \in A$ and $b_1, \dots, b_k \in BOUND$
3. $sbid^D : a_i \mapsto b_i$ is an injection
   with inverse $attr^D : b_i \mapsto a_i$

$STREAM$ is a set of names for temporal stream definitions. The schema S of $D = (n, S)$ is also denoted $\mathscr{S}(D)$.

4. OPERATORS RELOADED

TSA enhances the standard operators[2] of relational algebra with a mechanism for propagating constraints on temporal relations (TRFs) and progressing attributes (SBFs) to the schema of their output relation. So the interesting part of the following operator definitions is the definition of the schema for the result stream, not the definition of the result stream itself. Nevertheless we also give the definition for the output stream because first there hardly is *the one* representation of relational algebra, second we ourselves introduce two slight deviations from classical representations [2, 15] and third it seems convenient to have the definition of the result relations aside with the schema for these relations containing the propagated constraints. The two deviations from classical representations of relational algebra are as follows: First the basic TSA operators include a cross-product but no join operator. Second we separate the definition of new attribute from the projection operator and introduce the imbed operator ι for this purpose. Both deviations do not change the overall characteristics of the operators. In fact join, the classical projection and other frequently used operators like semi-join and anti-semi-join can be defined using the basic operators of TSA. However both deviations simplify the definition of constraint propagation as the operators become more orthogonal to each other, i. e., each performing a single task.

[17] An atom occurs with positive/negative polarity in C if it is found below an even/odd number of negations.

There are two operators that introduce new information on temporal relations namely selection σ and imbed ι. All other operators only combine and propagate the existing information in the schema of their input streams. The selection operator $\sigma[C]$ seeks to extracts the maximum temporal information in its selection condition C into a TRF C_{temp} and adds C_{temp} to the TRF from the schema of its input stream. Basically C_{temp} results from C by replacing all non-temporal atoms in such a way that the condition C is fulfilled as much as possible.

Definition 12. **(Selection σ)** Let E be a TSA expression with schema $\mathscr{S}(E) = (A, G, H)$ and let R be a temporal stream matching $\mathscr{S}(E)$ and C a condition with $attr(C) \subseteq A$.

$$\sigma[C](R) := \{r \in R \mid r \text{ satisfies } C\}$$
$$\mathscr{S}(\sigma[C](E)) := (A, (G \wedge C_{temp}), H)$$

where C_{temp} is the TRF that results from C when replacing every non-temporal atom by $\top$ if it occurs with positive polarity or by $\perp$ if it occurs with negative polarity.[17]

The imbed operator ι adds the definitions of relative timestamps to the TRF of its input stream.

Definition 13. **(Imbed ι)** Let E be a TSA expression with schema $\mathscr{S}(E) = (A, G, H)$ and let R be a temporal stream matching $\mathscr{S}(E)$. Let $a' \notin A$ be a new attribute, t the term defining a' and $a_1, \dots, a_k \in A$ the attributes occurring in t.

$$\iota[a' = t](R) := \{r' \in dom(A \cup \{a'\}) \mid \exists r \in R \text{ such that } r'(a) = r(a) \text{ for } a \in A \text{ and } r'(a') = f_t(r(a_1), \dots, r(a_k))\}$$
$$\mathscr{S}(\iota[a' = t](E)) := (A \cup \{a\}, G', H)$$
$$G' := \begin{cases} G \wedge (a' = t) & \text{if } t \text{ is a temporal term} \\ G & \text{else} \end{cases}$$

where $f_t : dom(a_1) \times \ldots \times dom(a_k) \to dom(a')$ is the function of the values of $a_1, \dots, a_k \in A$ defined by t. If t is a temporal term, then t defines a *relative timestamp*.

Both the projection and the grouping operators propagate the TRF from the schema of the input stream by replacing the occurrences of the discarded attributes by variables.

Definition 14. **(Projection π)** Let E be a TSA expression with schema $\mathscr{S}(E) = (A, G, H)$ and let R be a temporal stream which matches the schema of E and $A_1 \subseteq A$ the set of retained attributes.

$$\pi[A_1](R) := \{r' \in dom(A_1) \mid \exists r \in R \text{ such that } r'(a) = r(a) \text{ for } a \in A_1\}$$
$$\mathscr{S}(\pi[A_1](E)) := (A_1, \xi(G), \xi(H))$$

where ξ is an injective substitution of the attributes in $A \backslash A_1$ by variables that do not occur in G or H.

The simple definition of grouping might be surprising, as grouping is one of the "blocking" operator which are usually considered "problematic".

DEFINITION 15. **(Grouping γ)** Let E be a TSA expression with schema $\mathscr{S}(E) = (A, G, H)$ and let R be a temporal stream matching $\mathscr{S}(E)$ and $A_1 \subseteq A$ the set of grouping attributes. Let $a_1, \dots a_k \in A \setminus A_1$ and $a'_1, \dots, a'_k \notin A_1$ and $F_1, \dots, F_k$ aggregation functions like min, max, sum or avg.

$$\begin{aligned}
\gamma[A_1][a'_1 = F_1(a_1), \dots, a'_k = F_1(a_k)](R) := \\
\{r' \in dom(A_1 \cup \{a_1, \dots, a_k\}) \mid \exists r \in R \\
\text{such that } r'(a) = r(a) \text{ for } a \in A_1 \\
\text{and } r'(a'_i) = F_i((r_\gamma(a_i))_{r_\gamma \in R_r}) \quad \} \\
\mathscr{S}(\gamma[A_1][a'_1 = F_1(a_1), \dots, a'_k = F_1(a_k)](E)) := \\
(A_1, \xi(G), \xi(H))
\end{aligned}$$

where $R_r = \{r_\gamma \in R \mid r_\gamma(a) = r(a) \text{ for } a \in A_1\}$ and ξ is an injective substitution of the in attributes $A \setminus A_1$ by variables that do not occur in G or H.

The propagated TRF for the cross-product operator is basically the conjunction of the TRFs of the input streams.

DEFINITION 16. **(Cross Product $\times$)**[18] Let E_1 and E_2 be TSA expressions with schema $\mathscr{S}(E_1) = (A_1, G_1, H_1)$ and $\mathscr{S}(E_2) = (A_2, G_2, H_2)$ that have disjoint attributes ($A_1 \cap A_2 = \emptyset$) and let R_1 and R_2 be temporal streams matching $\mathscr{S}(E_1)$ and $\mathscr{S}(E_2)$ respectively.

$$\begin{aligned}
R_1 \times R_2 := \{r' \in dom(A_1 \cup A_2) \mid \exists r_1 \in R_1, r_2 \in R_2 \text{ with} \\
r'(a) = r_1(a) \text{ for } a \in A_1 \\
r'(a) = r_2(a) \text{ for } a \in A_2\} \\
\mathscr{S}(E_1 \times E_2) := (A_1 \cup A_2, \xi_1(G_1) \wedge \xi_2(G_2), \xi_1(H_1) \wedge \xi_2(H_2))
\end{aligned}$$

where ξ_1 and ξ_2 are injective substitutions, such that $\xi_1(G_1)$, $\xi_1(H_1)$ and $\xi_2(G_2)$, $\xi_2(H_2)$ use disjoint sets of variables.

The propagated TRF for the union operator is basically the disjunction of the TRFs of the input streams. Some details of the definition are of very technical nature,[11] particularly G''_1 and G''_2.

DEFINITION 17. **(Union $\cup$)**[18] Let E_1 and E_2 be TSA expressions with equal attributes, schema $\mathscr{S}(E_1) = (A, G_1, H_1)$ and $\mathscr{S}(E_2) = (A, G_2, H_2)$ and let R_1 and R_2 be temporal streams matching $\mathscr{S}(E_1)$ and $\mathscr{S}(E_2)$ respectively.

$$\begin{aligned}
R_1 \cup R_2 &:= \{r \in dom(A) \mid \text{with } r \in R_1 \text{ or } r \in R_2\} \\
\mathscr{S}(E_1 \cup E_2) &:= (A, G', (\xi_1(H_1) \wedge \xi_2(H_2))) \\
G' &:= (G'_1 \wedge G''_1 \wedge \xi_1(G_1)) \vee (G'_2 \wedge G''_2 \wedge \xi_2(G_2)) \\
G'_1 &:= (\xi_1(p_1) = p_1) \wedge \dots \wedge (\xi_1(p_k) = p_k) \\
G''_1 &:= \bigwedge_{v \in attr(H_2) \cup vars(H_2)} ignore(\xi_2(v)) \\
G'_2 &:= (\xi_2(p_1) = p_1) \wedge \dots \wedge (\xi_2(p_k) = p_k) \\
G''_2 &:= \bigwedge_{v \in attr(H_1) \cup vars(H_1)} ignore(\xi_1(v))
\end{aligned}$$

where ξ_1 and ξ_2 are injective substitutions, that replace all attributes in A by variables and substitute variables by others such that $\xi_1(v) \neq \xi_2(w)$ for any two attributes or variables v and w occurring in A, G_1, G_2, H_1 or H_2.

The propagated TRF for the set-difference operator is very technical nature.[11] The challenge is that the temporal relations between the *attributes* of the result stream are only determined by the temporal relations of the first input stream, i. e. by G_1. Actually the propagated TRF G' is equivalent to G_1 with respect to the temporal relations between the *attributes* of the result stream. However G' also needs to propagate the information in the TRF G_2 of the second input. Otherwise the relations between the progressing sets of attributes for the result stream and the progressing sets of attributes for the second input stream get lost.

DEFINITION 18. **(Set Difference $\setminus$)** Let E_1 and E_2 be TSA expressions with equal attributes, schema $\mathscr{S}(E_1) = (A, G_1, H_1)$ and $\mathscr{S}(E_2) = (A, G_2, H_2)$ and let R_1 and R_2 be temporal streams matching $\mathscr{S}(E_1)$ and $\mathscr{S}(E_2)$ respectively.

$$\begin{aligned}
R_1 \setminus R_2 &:= \{r_1 \in R_1 \mid \neg \exists r_2 \in R_2 \text{ with } r_1 = r_2\} \\
\mathscr{S}(E_1 \setminus E_2) &:= (A, G', (H_1 \wedge \xi_2(H_2))) \\
G' &:= (G_1 \wedge G'_1) \vee (G_1 \wedge G'_2 \wedge \xi_2(G_2))) \\
G'_1 &:= \bigwedge_{v \in attr(H_2) \cup vars(H_2)} ignore(\xi_2(v)) \\
G'_2 &:= (\xi_2(p_1) = p_1) \wedge \dots \wedge (\xi_2(p_k) = p_k)
\end{aligned}$$

where ξ_2 is an injective substitution for attributes and variables, such that the substituted variants $\xi_2(G_2)$, $\xi_2(H_2)$ and G_1, H_1 use disjoint sets of variables.

DEFINITION 19. **(TSA Query)**

1. A TSA *query* is a pair $q = (D, E)$ such that E is a TSA expression, D is a temporal stream definition and the schema of E and D have the same set of attributes.
2. q is *valid* iff the schema $\mathscr{S}(E) = (A, G_E, H_E)$ of E matches the schema $\mathscr{S}(D) = (A, G_D, H_D)$ of D, i. e. G_E implies[19] G_D and all progressing sets of attributes with respect to G_D and H_D are progressing sets of attributes with respect to G_E and H_E.

A TSA query assigns a TSA expressions to the temporal stream it contributes to. The most important property of *valid* TSA queries is, that they are non-blocking, i. e. can be evaluated incrementally.

PROPOSITION 20. **(Non-Blocking Queries)** Let $q = (D, E)$ be a TSA query with schema $\mathscr{S}(D) = (A, G_D, H_D)$ and let $D_1, \dots, D_k$ be the temporal stream definitions occurring in E and $R_1, \dots, R_k$ temporal streams matching the schema $\mathscr{S}(D_1), \dots, \mathscr{S}(D_k)$ respectively.

If q is *valid* then q is *non-blocking*. This means and any limit $s \in \mathbb{Q}$ for a progressing attribute[9] $a \in A$ of $\mathscr{S}(D)$, there is a point in time $p \in \mathbb{Q}$ such that for every point in time $p' \geq q$ the prefix of the result stream up to s is fully determined by the prefixes of the input streams at time p'.

$$\{r \in E(R_1, \dots, R_k) \mid r(a) \leq s\} = \\ \{r \in E(R_1^{p'}, \dots, R_k^{p'}) \mid r(a) \leq s\}$$

PROOF (SKETCH). Let $\mathscr{S}(E) = (A, G_E, H_E)$. Without loss of generality one may assume that the temporal stream definitions $D_1, \dots, D_k$ use different stream bound identifiers, as Proposition 20 and none of its indirectly referred Definitions depend on the actual names of stream bounds.

As q is a *valid* query, any progressing attribute[9] $a \in A$ for $\mathscr{S}(D)$ is a progressing attribute[9] for $\mathscr{S}(E)$.

Let G_E^{norm} be the normalized form of G_E (see Definition 4) and the DNF of G_E^{norm} be $dnf(G_E^{norm}) = C_1 \vee \dots \vee C_l$, where $C_1, \dots, C_l$ are conjunctions of normalized atomic TRFs.

For each D_i, $1 \leq i \leq k$ and each C_j there must exist at least one[20] atomic SBF $bounded(v_{i,j}, b_{i,j})$ in H_E[21] where $v_{i,j} \leq a + dist_{C_j}(v_{i,j}, a)$ and $b_{i,j}$ belongs to D_i.

[18] The definition can easily be extended to a n-ary operator.

[19] Can be checked if right side does not contain variables. By definition G_D and H_D do not contain variables.

[20] If D_i describes a static relation this does not hold. But in that case there is nothing to show. One can choose $l_i = 0$.

[21] Note that H_E is in CNF with exactly one clause per $D_1, \dots, D_k$ (Proposition 21)

However $v_{i,j}$ is a renamed version of the progressing attribute[9] $a_{i,j} := attr^{D_i}(b_{i,j})$ associated to $b_{i,j}$ in D_i. Thus if $a \leq s$ in the result stream then $a_{i,j} \leq s_{i,j} := s - dist_{C_j}(v_{i,j}, a)$ in the input stream D_i, at least for "case" C_j of G_H^{norm}.

Proposition 22 allows to apply the formal results from the definitions D_i to the actual relations R_i. Therefore

$$\{r \in E(R_1, \ldots, R_k) \mid r(a) \leq s\} = \{r \in E(P_1, \ldots, P_k) \mid r(a) \leq s\}$$

where

$$P_i := \bigcup_{1 \leq j \leq l} \{r_i \in R_i \mid r_i(a_{i,j}) \leq s_{i,j}\}$$

As $a_{i,j}$ is a progressing attribute of R_i there is a point in time $p_{i,j}$ such that $\{r_i \in R_i \mid r_i(a_{i,j}) \leq s_{i,j}\} \subseteq R_i^{p_{i,j}}$. Therefore with $p := \max_{\substack{1 \leq i \leq k \\ 1 \leq j \leq l}} p_{i,j}$ and $R_i^p \subseteq R_i^{p'}$ iff $p \leq p'$

$$\{r \in E(R_1, \ldots, R_k) \mid r(a) \leq s\} = \{r \in E(R_1^{p'}, \ldots, R_k^{p'}) \mid r(a) \leq s\} \quad \square$$

The proof of Proposition 20. uses the following two auxiliary proposition on the structure of propagated SBFs and on the well-definedness of the propagation mechanism:

Proposition 21. **(Simple SBFs)** For any TSA expression E with schema $\mathscr{S}(E) = (A, G_E, H_E)$ the SBF H_E is in conjunctive normal form (CNF).

Proof. The proposition holds for temporal stream definitions (Definition 11). For each TSA operator the SBF of the composed expression is a conjunction of the SBFs of the subexpressions. $\square$

Proposition 22. **(Well-Definedness)** Let E be a TSA expression with the *valid* schema $\mathscr{S}(E)$ where $D_1, \ldots, D_k$ are the temporal stream definitions occurring in E.

If $R_1, \ldots, R_k$ are temporal streams matching the schema of $D_1, \ldots, D_k$ respectively then the output $E(R_1, \ldots, R_k)$ is a temporal stream and matches the schema $\mathscr{S}(E)$.

Proof (Sketch). The proof is straight-forward by induction over the structure of E. $\square$

Proposition 23. **(Relational Completeness)** TSA is *relational complete* [10] for finite (non-stream) relations.[22]

Proof. On finite non-stream relations, TSA is equivalent to relational algebra. $\square$

5. TSA BY EXAMPLE

This section illustrates how frequent types of queries can be expressed using the operators of TSA. As to keep the TSA expressions more compact and readable, operators like rename δ, join $\bowtie$, semi-join $\ltimes$ and anti-semi-join $\overline{\ltimes}$ are used in the examples. These operators can easily be composed from the basic TSA operators. All shown queries are valid TSA queries (Definition 19) and all but the query for frames without fragments can be garbage collected solely based on an analysis of temporal relations (see end of Section 6).

Stream and Static Data. The following examples shows a rule that takes raw events from temperature sensors *TRaw* and supplements them with the location (*area*) of the sensors stored in static relation *TLoc*.

[22] The term "relational complete" compares the expressive power of some formalism for querying *finite relations* to the expressive power of relational algebra. A generalization to temporal streams is, if at all possible, non-trivial.

$\mathscr{S}(TRaw) = (\{sid, time, temp\}, \top, bounded(TRaw{:}time, time))$
$\mathscr{S}(TLoc) = (\{sid, area\}, \top, \top)$
$\mathscr{S}(Temp) = (\{sid, time, area, temp\}, \top, bounded(TRaw{:}time, time))$

$$
\begin{array}{l}
Temp \leftarrow \\
\quad \delta[t.* \rightarrow *,\ l.area \rightarrow area](\\
\quad\quad \pi[t.sid,\ t.time,\ t.temp,\ l.area](\\
\quad\quad\quad \sigma[t.sid = l.sid] \\
\quad\quad\quad\quad \delta[* \rightarrow t.*](TRaw) \times \delta[* \rightarrow l.*](TLoc) \\
\quad)))
\end{array}
$$

Tumbling Windows. The following rule is a typical example for tumbling windows. The rule computes the average speed of air movement measured by a certain airflow sensor within the last 20 seconds. This information is required as input for the simulation of smoke propagation.

$\mathscr{S}(AFRaw) = (\{sid, time, speed\}, \top, bounded(AFRaw{:}time, time))$
$\mathscr{S}(Airflow) = (\{sid, time, speed\}, \top, bounded(Airflow{:}time, time))$

$$
\begin{array}{l}
Airflow \leftarrow \\
\quad \delta[time20sec \rightarrow time,\ avgSpeed \rightarrow speed](\\
\quad\quad \gamma[sid,\ time20sec][avgSpeed = avg(speed)](\\
\quad\quad\quad \iota[time20sec = ceil_{20sec}(time)](\\
\quad\quad\quad\quad AFRaw \\
\quad)))
\end{array}
$$

Lazy Aligned Sliding Windows. The following example shows a sliding window, which is only evaluated when actually needed (lazy), and which is aligned to the event initiating the evaluation. The implemented query reads as follows: From the start to one hour after the end of an fire compute the average smoke concentration in room in the last 10 seconds with updates every second. The implementation of the windows uses the safe, i. e., non-blocking form of recursion provided by TSA.[23]

$$
\begin{array}{rl}
\mathscr{S}(Smoke) = & (\{time, area, conz\}, \top, \\
& bounded(Smoke{:}time, time) \\
\mathscr{S}(FireStart) = & (\{fid, time\}, \top, \\
& bounded(FireStart{:}time, time) \\
\mathscr{S}(FireEnd) = & (\{fid, time\}, \top, \\
& bounded(FireEnd{:}time, time) \\
\mathscr{S}(Windows) = & (\{start, end\}, start = end - 10sec, \\
& bounded(Windows{:}start, start) \vee \\
& bounded(Windows{:}end, end)\}) \\
\mathscr{S}(AvgSmoke) = & (\{time, area, acgConz\}, \top, \\
& bounded(AvgSmoke{:}time, time)
\end{array}
$$

$$
\begin{array}{l}
AvgSmoke \leftarrow \\
\quad \delta[s.area \rightarrow area,\ w.end \rightarrow time](\\
\quad\quad \gamma[s.area,\ w.end][avgConz = avg(conz)](\\
\quad\quad\quad \sigma[w.start \leq s.time \leq w.end](\\
\quad\quad\quad\quad \delta[* \rightarrow s.*](Smoke) \times \delta[* \rightarrow w.*](Windows) \\
\quad))) \\
Windows \leftarrow \\
\quad \pi[start,\ end](\\
\quad\quad \iota[start = end - 10sec](\\
\quad\quad\quad \delta[time \rightarrow end](\\
\quad\quad\quad\quad FireStart \\
\quad))) \\
Windows \leftarrow \\
\quad \delta[nstart \rightarrow start,\ nend \rightarrow end](\\
\quad\quad \pi[nstart,\ nend](\\
\quad\quad\quad \iota[nstart = w.start + 1sec, nend = w.end + 1sec](\\
\quad\quad\quad\quad \overline{\ltimes}\ [w.start \leq f.time + 1h < w.end](\\
\quad\quad\quad\quad\quad \delta[* \rightarrow w.*](Windows), \\
\quad\quad\quad\quad\quad \delta[* \rightarrow f.*](FireEnd) \\
\quad\quad\quad\quad) \\
\quad)))
\end{array}
$$

Frames. The following query implements a frame [23] starting at the beginning of a fire, lasting till the end of that fire and computing the maximum smoke concentration in every area during that period when the duration of the fire is at least one minute.

[23] Details on recursion are out of the scope of this article.

$$\begin{array}{l}
\mathscr{S}(\mathit{MaxSmoke}) = (\{\mathit{start}, \mathit{end}, \mathit{area}, \mathit{maxConz}\}, \\
\qquad\qquad\qquad\qquad \mathit{end} - \mathit{start} \geq 1\mathit{min}, \\
\qquad\qquad\qquad\qquad \mathit{bounded}(\mathit{MaxSmoke}{:}\mathit{end}, \mathit{end}) \\
\mathit{MaxSmoke} \leftarrow \\
\quad \delta[\mathit{fs.time} \to \mathit{start},\ \mathit{fe.time} \to \mathit{end}, \\
\quad\ \ \mathit{fs.fid} \to \mathit{fid},\ \mathit{s.area} \to \mathit{area}](\\
\quad\ \ \gamma[\mathit{fs.time},\ \mathit{fe.time},\ \mathit{s.area}][\mathit{maxConz} = \mathit{max}(\mathit{conz})](\\
\quad\ \ \ \ \sigma[\mathit{fs.time} \leq \mathit{s.time} \leq \mathit{fe.time}, \\
\quad\ \ \ \ \ \ \mathit{fe.time} - \mathit{fs.time} \geq 1\mathit{min},\ \mathit{fs.fid} = \mathit{fe.fid}](\\
\quad\ \ \ \ \ \ \delta[* \to \mathit{fs}.*](\mathit{FireStart}) \times \\
\quad\ \ \ \ \ \ \delta[* \to \mathit{s}.*](\mathit{Smoke}) \times \\
\quad\ \ \ \ \ \ \delta[* \to \mathit{fe}.*](\mathit{FireEnd}) \\
\quad)))
\end{array}$$

Frame Fragments. The following example implements frame fragments [23] for the preceding query. The first frame fragment is produced after one minute when it is known that a frame exists. Each minute another frame fragment is produced until the end of the frame, i. e., the end of the fire is detected. Then the final result for the frame is produced. Frame fragments increase efficiency, as smoke events only need to be stored for and averages only need to be computed over a period of one minute. The implementation uses the safe, i. e., non-blocking form of recursion provided by TSA.[23]

$$\begin{array}{l}
\mathscr{S}(\mathit{MaxSmokeFrag}) = (\{\mathit{start}, \mathit{end}, \mathit{area}, \mathit{maxConz}\}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{end} - \mathit{start} \geq 1\mathit{min}, \\
\qquad\qquad\qquad\qquad\qquad \mathit{bounded}(\mathit{MaxSmokeFrag}{:}\mathit{end}, \mathit{end})\}) \\
\mathit{MaxSmokeFrag} \leftarrow \\
\quad \iota[\mathit{end} = \mathit{start} + 1\mathit{min}](\\
\quad\ \ \delta[\mathit{fs.time} \to \mathit{start},\ \mathit{s.area} \to \mathit{area}](\\
\quad\ \ \ \ \gamma[\mathit{fs.time}, \mathit{s.area}][\mathit{maxConz} = \mathit{max}(\mathit{conz})](\\
\quad\ \ \ \ \ \ \overline{\ltimes}\ [\mathit{fs.time} \leq \mathit{fe.time} < \mathit{fs.time} + 1\mathit{min}](\\
\quad\ \ \ \ \ \ \ \ \sigma[\mathit{fs.time} \leq \mathit{s.time} < \mathit{fs.time} + 1\mathit{min}](\\
\quad\ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{fs}.*](\mathit{FireStart}) \times \\
\quad\ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{s}.*](\mathit{Smoke}) \\
\quad\ \ \ \ \ \ \ \ ), \\
\quad\ \ \ \ \ \ \ \ \delta[* \to \mathit{fe}.*](\mathit{FireEnd}) \\
\quad)))) \\
\mathit{MaxSmokeFrag} \leftarrow \\
\quad \pi[\mathit{start}, \mathit{end}, \mathit{area}, \mathit{maxConz}] \\
\quad\ \ \delta[\mathit{fs.start} \to \mathit{start},\ \mathit{fs.area} \to \mathit{area}](\\
\quad\ \ \ \ \iota[\mathit{end} = \mathit{fs.end} + 1\mathit{min}, \\
\quad\ \ \ \ \ \ \mathit{maxConz} = \mathit{max}(\mathit{fs.maxConz}, \mathit{NMC})](\\
\quad\ \ \ \ \ \ \gamma[\mathit{fs.start},\ \mathit{fs.end},\ \mathit{fs.area}][\mathit{NMC} = \mathit{max}(\mathit{conz})](\\
\quad\ \ \ \ \ \ \ \ \overline{\ltimes}\ [\mathit{fs.end} \leq \mathit{fe.time} < \mathit{fs.end} + 1\mathit{min},](\\
\quad\ \ \ \ \ \ \ \ \ \ \sigma[\mathit{fs.end} \leq \mathit{s.time} < \mathit{fs.end} + 1\mathit{min}, \\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \mathit{fs.area} = \mathit{s.area}](\\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{fs}.*](\mathit{MaxSmokeFrag}) \times \\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{s}.*](\mathit{Smoke}) \\
\quad\ \ \ \ \ \ \ \ \ \ ), \\
\quad\ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{fe}.*](\mathit{FireEnd}) \\
\quad))))) \\
\mathit{MaxSmoke} \leftarrow \\
\quad \pi[\mathit{start}, \mathit{end}, \mathit{area}, \mathit{maxConz}] \\
\quad\ \ \delta[\mathit{fs.start} \to \mathit{start},\ \mathit{fs.area} \to \mathit{area}](\\
\quad\ \ \ \ \iota[\mathit{end} = \mathit{fe.time} \\
\quad\ \ \ \ \ \ \mathit{maxConz} = \mathit{max}(\mathit{fs.maxConz}, \mathit{NMC})](\\
\quad\ \ \ \ \ \ \gamma[\mathit{fs.start},\ \mathit{fe.time},\ \mathit{s.area}][\mathit{NMC} = \mathit{max}(\mathit{conz})](\\
\quad\ \ \ \ \ \ \ \ \sigma[\mathit{fs.end} \leq \mathit{s.time} < \mathit{fe.time}, \\
\quad\ \ \ \ \ \ \ \ \ \ \mathit{fs.area} = \mathit{fs.area} \\
\quad\ \ \ \ \ \ \ \ \ \ \mathit{fs.end} \leq \mathit{fe.time} < \mathit{fs.end} + 1\mathit{min}](\\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{fs}.*](\mathit{MaxSmokeFrag}) \times \\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{s}.*](\mathit{Smoke}) \times \\
\quad\ \ \ \ \ \ \ \ \ \ \ \ \delta[* \to \mathit{fe}.*](\mathit{FireEnd}) \\
\quad)))))
\end{array}$$

6. INCREMENTAL EVALUATION

The incremental evaluation is obviously crucial for TSA. It allows to derive results continuously as the data arrives on the stream. The incremental evaluation bases on the important property of valid TSA queries shown in Proposition 20: Any prefix of the result stream of a query q depends only on finite prefixes of the input streams of q.

The major challenge for the incremental evaluation of TSA queries is to pass only those results to the output that are complete or stable, i. e. don't change on the arrival of further input data. This is realized by *pass conditions* (Definition 27) which accept only those result tuples that can safely be output based on the current progress of the input streams. Pass conditions are very similar to the "pass invariants" of [25]. *Pass formulas* (Definition 24) are parametrized versions of the pass conditions which can be derived at compile-time. The actual pass condition is obtained from the pass formula by replacing the parameters $b \in \mathit{BOUND}$ (formally called "stream bound identifiers") by their so-called *progress values* (Definition 26). Progress values basically tell about the current progress of a data stream with respect to its different progressing attributes.[9] Progress values are similar to linear punctuations [21] for the output stream of a query with the main difference that progress values are not imbedded into the output stream.

Definition 24. **(Pass Formulas)** Let H be a SBF. The *pass formula* $\triangle H$ to H is defined recursively:

$$\begin{array}{ll}
\triangle\top & = \top \\
\triangle\ \mathit{bounded}(p, b) & = p \leq b \\
\triangle(H_1 \wedge \ldots \wedge H_k) & = \triangle H_1 \wedge \ldots \wedge \triangle H_k \\
\triangle(H_1 \vee \ldots \vee H_k) & = \triangle H_1 \vee \ldots \vee \triangle H_k
\end{array}$$

where $p \in \mathit{ATTR} \cup \mathit{VAR}$ and $b \in \mathit{BOUND}$. $\triangle H$ results from H by replacing each atom $\mathit{bounded}(p, b)$ in H by $p \leq b$.

When instantiating a pass formula to obtain the pass condition for the current increment computation of a query those progress values are used that will hold after the current increment computation has finished. The progress for the output stream that is achievable with the current increment computation depends on the progress of the input streams at the beginning of the increment computation, more precisely the progress values for the output stream at the *end* of the increment computation depend on the progress values of the input streams at the *beginning of the increment computation.* The actual function that computes the progress values for the output stream based on the progress values of the input streams is called *propagation function* (Definition 25) and is derived at compile-time. Propagation functions are closely related to the propagation invariants of [25].

Definition 25. **(Propagation Functions)**[24]
The *propagation function* for a TSA query $q = (D, E)$ with $\mathscr{S}(D) = (A, G_D, H_D)$ and $\mathscr{S}(E) = (A, G_E, H_E)$, and a stream bound identifier $b \in \mathit{BOUND}$ with corresponding attribute $a = \mathit{attr}^D(b_D) \in A$ (Definition 11) for which the atom $\mathit{bounded}(a, b)$ is contained in H_D, is

$$\begin{array}{ll}
f_{b,q} & = f_{b_D, G_E, \triangle H_E} \\
f_{b,G_E,\triangle H_E} & = \min\limits_{C_E \in \mathit{dnf}(G_E^{norm})} f_{b,C_E,\triangle H_E} \\
f_{b,C_E,\top} & = +\infty
\end{array}$$

[24] Without loss of generality the definition assumes that all temporal stream definitions of a TSA program use different stream bound identifiers $b \in \mathit{BOUND}$. This can easily be realized if $\mathit{BOUND} = \mathit{STREAM} \times \mathit{ATTR}$ and for temporal stream definition $D = (n, S)$ and progressing attribute[9] p of S the stream bound identifier $b = b^D(p)$ corresponding to p in D has the form $b = n : p$.

$$f_{b,C_E,(v \leq b_E)} = \begin{cases} +\infty & \text{if } dist_{C_E}(a,v) = -\infty \\ -\infty & \text{if } dist_{C_E}(a,v) = +\infty \\ b \leq b_E - dist_{C_E}(a,v) & \text{else} \end{cases}$$
$$f_{b,C_E,(\triangle H_{E_1} \wedge \ldots \wedge \triangle H_{E_k})} = \min(f_{b,C_E,\triangle H_{E_1}}, \ldots, f_{b,C,\triangle H_{E_k}})$$
$$f_{b,C_E,(\triangle H_{E_1} \vee \ldots \vee \triangle H_{E_k})} = \max(f_{b,C_E,\triangle H_{E_1}}, \ldots, f_{b,C,\triangle H_{E_k}})$$

where $v \in ATTR \cup VAR$, $b_E \in BOUND$ and C_E is a conjunction of atoms of the form $v \leq b_E$ and $ignore(v)$.

The idea is, to enforce $v \leq b_E$ in the input relation using a and b_D from the output relation and the inequation

$$v \leq a + dist_{C_E}(a,v) \leq b + dist_{C_E}(a,v) \overset{!}{\leq} (b_E - dist_{C_E}(a,v)) + dist_{C_E}(a,v) \leq b_E$$

The minimum in the first case results from the fact that H_E must hold in each case of G_E (compare with Definition 6).

The propagation function $f_{b,q}$ can be normalized to be $+\infty$ or $-\infty$ or not to contain $+\infty$ or $-\infty$ at all. If $f_{b,q} = -\infty$, then a is not a progressing attribute[9] with respect to G_E and H_E. This will raise a compile error, as it actually means that the query is invalid (Definition 19). If $f_{b,q} = +\infty$ then G_E and H_E belong either to a query that refers only to static relations or to a query that has an empty output stream as it imposes inconsistent temporal relations.

As already mentioned progress values tell about the current progress of a data stream with respect to its different progressing attributes.[9] The progress values of the output stream of a query are computed by applying the corresponding propagation functions to the progress values of the input streams of the query. The progress value for a derived data stream is the minimum progress value of the output streams of all queries contributing to the derived data stream.

Definition 26. **(Progress Values)**[24]
Let P be a set of TSA Queries. For a TSA Query $q = (D, E)$ with $\mathscr{S}(D) = (A, G_D, H_D)$ let $s_q^i \in \mathbb{Q}$ and $e_q^i \in \mathbb{Q}$ denote the start and end time of the ith ($i \geq 1$) increment computation for q. The following is defined simultaneously:

1. Progress value of b for query q at the ith increment computation for q and $i \geq 0$
   $prop_q^0(b) = -\infty$ [25]
   $prop_q^i(b) = f_{b,q}(prop_{s_q^i}(b_1), \ldots, prop_{s_q^i}(b_l))$
   where b occurs in H_D and $f_{b,q}$ is the propagation function corresponding to q and b (Definition 25) and $b_1, \ldots, b_l$ are the stream bound identifiers occurring in $f_{b,q}$.
2. Progress value of b for query q at time $p \in \mathbb{Q}$
   $$prop_q^p(b) = \begin{cases} prop_q^0(b) \text{ for } p < e_q^1 \\ prop_q^i(b) \text{ for } e_q^i \leq p < e_q^{i+1} \end{cases}$$
3. Value of b for temporal stream definition D at time p
   $$prop_D^p(b) = \min_{\substack{q' \in P \\ \text{out}(q') = D}} \{prop_{q'}^p(b)\}$$
   The progress value $prop_D^p(b)$ for input streams must be provided form outside.[26]
4. Progress value of b at time p
   $prop^p(b) = prop_{D_b}^p(b)$
   where D_b is the temporal stream definition corresponding to b.[24]

The definition is well defined as $s_q^i < e_q^i$ for $i \geq 1$.

For hierarchical programs the incremental evaluation also works with conservative approximations of the progress values.[27] This is particularly useful for a parallel or even distributed evaluation of a TSA program, as the progress values do not need to be perfectly synchronized. Furthermore conservative approximations for the propagation functions $f_{b,q}$ could be used. The propagation functions $f_{b,q}$ of Definition 25 are optimal with respect to the achievable progress within one step, however approximate,i. e. simpler versions of the functions, may help to reduce the computational overhead that is potentially introduced by the computation of the progress values.

The ith increment expression, i. e.the expression that is actually executed for performing the ith increment computation of a query, consists of the pass condition (and the negated pass condition of the last increment computation) placed on top of the original query expression.[28]

Definition 27. **(Increment Expression)**
Let $q = (D, E)$ be a *valid* TSA query with output schema $\mathscr{S}(D) = (A, G_D, H_D)$ and $\triangle H_D = a_1 \leq b_1 \vee \ldots \vee a_k \leq b_k$.

1. The *pass condition* for $i \geq 0$ is:
   $\triangle H_D^i = a_1 \leq prop_q^i(b_1) \vee \ldots \vee a_k \leq prop_q^i(b_k)$
2. The *increment expression* for $i \geq 1$ is:
   $\triangle E^i = \sigma[\triangle H_D^i \wedge \neg \triangle H_D^{i-1}](E)$

The increment expression for a TSA query can be seen as parametrized relational algebra expression as it is only evaluated against finite relations (the currently buffered part of the input streams) and the only changes from one increment computation to the other are the progress values within the pass conditions. This is very useful as it avoids to compile each of the increment expressions at the run-time of a TSA program. Instead the parametrized version of the increment expression can be compiled only once. Furthermore an efficient implementation of relational algebra on finite standing relations can easily be enhanced to an implementation of TSA, i. e.relational algebra on data streams .

Proposition 28. **(Correctness)** The consecutive execution of the increment expressions $\triangle E_1, \triangle E_2, \triangle E_3, \ldots$ for a TSA query $q = (D, E)$ yields the same result as if E had been applied to the whole stream at once.

Proof (Sketch). The proof is similar to the proof of Proposition 20. The proof of Proposition 20 just uses the fact that there must exist some "progress values". Thus the proof of Proposition 20 basically has to show that the "progress values" required by the proof of Proposition 20 are actually just those progress values computed by the propagation functions of Definition 25. This is the case as Definition 25 is well aligned with Definition 7 of progressing sets of attributes and Definition 5 of temporal distance. □

[25] For recursive programs the initial values have to chosen more carefully. This is out of the scope of this article.

[26] See [25, 20] for a discussion on how obtain punctuations for input streams.

[27] In the case of a recursive program this may not hold for the progress values of queries and temporal stream definitions in a strongly connected component of the program. It still holds for the hierarchical parts of the recursive program.

[28] The query expression can be enhaced with relevance filters that accept only those tuples from the input streams that could contribute to new tuples in the result stream. This keeps intermediate results small and avoids recomputations.

Relevance & Garbage Collection

The incremental evaluation of TSA uses the fact that upper bounds to the result stream of a valid TSA query impose upper bounds to required parts of the input streams. For many TSA queries the same also holds for lower bounds: Lower bounds to the result stream impose lower bounds to the required parts of the input streams. This observation can be used to determine the relevance of tuples in the input streams of a query for further results of the query. More precisely, if the result stream of a query progresses beyond a certain point, indicated by the corresponding progress values (Definition 26), then for each input stream there exist so-called *keep values* such that only tuples with timestamps greater than these keep values can contribute to further results. Analog to the propagation functions (Definition 25) for the incremental evaluation, we can derive so-called *keep functions* from the temporal relations specified in the query. The keep functions are able to compute the current keep values for the input streams of a query based on the current progress values of the result stream of a query. The details are out of the scope of this article.

7. RELATED WORK

Windows. Windows appear in the form of tumbling, sliding or landmark windows [4, 9, 1], generalized window operators like in the WID [20], band predicates [21, 11] and validity intervals [19]. As shown in [23, 16] and in this article, new applications require queries that cannot be conveniently expressed using those data independent windows.

Pattern Matching. The (evaluation) semantics of pattern matching approaches like CEDR[6], SASE+[3], AFAs[8] and CAYUGA [17] usually bases on some automaton model. Inherently their semantics strongly relies on some order on the processed tuples or events. The approach is very well suited for applications like in financial markets where queries are formulated against a single timeline, queries with a "triggering" event at the beginning of the queried period[29] and extremely low response times are required. However it is unclear how automaton-based approaches could be extended to queries referring to multiple timelines, as in the presence of multiple timelines the order of the events is usually unclear.

Frames & Frame Segments. The motivation of frames [23] is similar to the idea of "event-controlled aggregation". Generally the borders of a frames are determined by means of changes in the arriving data. In the case of T+D frames the value of some attribute must be above some threshold for all tuples within the borders of a frame. The "Thresshold-Frame" and "FillFrame" operators proposed for T+D frames can be expressed by a composition of basic TSA operators. As frames may have arbitrary duration [23] introduces frame fragments for first detecting frames early, i. e. when their existence but not their end is known, and second enabling precomputations of subsequent operators as to reduce memory requirements and avoid workload bursts at the end of a frame. The intentions of frame fragments can partially be realized using the safe form recursion supported by TSA.[23]

Predicate Windows. Predicate windows are an approach towards data dependent windows. Predicate windows are motivated by the perspective that a window defines a set of currently valid tuples where tuples are entering into and expiring from this set based on a predicate assigned to the window. The approach apparently consists of two components: First predicate windows are used to interpret a data stream as a sequence of updates and deletes to a set of currently valid tuples and second a streamed version, i. e. no materialization of result relations, of a view maintenance algorithm propagates changes through subsequent operator.

Punctuations & Out-of-order Processing. Punctuations have been proposed first in [25] as an alternative to windows for unblocking operators and for limiting the state that an operators has to maintain. The WID approach in [20] showed that punctuation can be used to handle disordered streams and introduced operators that hardly need to reorder tuples for processing. The architecture for out-of-order processing (OOP) proposed in [21] bases on the WID approach and uses linear punctuations for unblocking operators and propagating stream progress. [21] shows that out-of-order processing can clearly out-perform in-order processing The linear punctuations of OOP are almost equivalent to the progress values used in the out-of-order incremental evaluation of TSA expressions. DataCell [22] also performs out-of-order processing however using (primarily tuple-based) windows instead of punctuations.

CERA. The Complex Event Relational Algebra (CERA) [12, 13] serves as operational semantics for the high-level complex event processing language XChange^{EQ}. CERA uses an analysis of temporal relations for garbage collection but not for incremental evaluation like TSA. CERA has some capabilities for the definition of relative timestamps but the timestamps of composite events are mostly predefined. Basic temporal relations can be combined using conjunctions, disjunctions and negation are missing, though. To some extend CERA can express windows with variable size. CERA allows simultaneous events, i. e. does not need a strict order on events, however the evaluation CERA needs to processes the sets of simultaneous events in temporal order. Consequently CERA only support a single timeline.

8. CONCLUSION & FUTURE WORK

We introduced a common data model for data streams and static relations and define Temporal Stream Algebra (TSA) enhancing the standard operators[2] of relational algebra by an mechanism for propagating constraints on temporal relations between attributes and on the correlation of attributes to the stream progress within the stream schema. We presented an analysis for the propagated constraints that can determine the validity of TSA queries and derives functions that define first when results can be passed on and second how information on stream progress is propagated and third which data needs to be kept. Based on these functions we describe an bulk-wise and out-of-order evaluation for TSA queries. In this way TSA can do without windows with fixed, data independent, sizes.[30] Outstanding features of TSA are the support of event-controlled aggregation, user-defined timestamps for composite events and multiple timelines and the smooth integration of static relations.

TSA serves as operational semantics of the event, state and action language Dura [28, 26, 27]. A prototype implementation of TSA and Dura on top of MonetDB has been

[29] Queries with negation or aggregation to the past, like "Return event B if there was no event A in the past 5 minutes", where the "trigger" for the query is at the end of the queried period, are problematic for automaton-based approches.

[30] Nevertheless such windows can be expressed when needed.

completed and used in prototypes of emergency management applications for metro networks, airports and powergrids[31, 27]. The prototype of TSA implements garbage collection based on the analysis of temporal relations. Currently we work on further improvements on the garbage collection and on the optimization of the incremental evaluation using so-called relevance filters.

The prototype currently supports a limited but safe, i. e. non-blocking, form of "recursion to the future".[31] We would like to examine other forms of recursion for example how the Flying Fixed-Point operator [23] could be generalized to strongly connected components in a recursive TSA program.

The incremental evaluation of TSA performs a bulk-wise out-of-order processing of events. We expect that this contributes to a high throughput, though it probably does not achieve minimum response times for single events, and helps to cope with peaks in the event load. We currently implement the Linear Road Benchmark [5] as to verify these claims. Moreover incremental evaluation enables an asynchronous query processing. We are very interested whether this allows an easy distributed processing.

The combination of asynchronous and bulk-wise processing opens a number of interesting questions with regards to scheduling. Though any fair execution sequence for the increment expressions of the queries will yield the correct results, scheduling affects the efficiency and the response-time of the evaluation. Determining a good or even optimal sequence is another of our current research issues. The same holds for a situation dependent query prioritization.

9. ACKNOWLEDGMENTS

The work presented in this article is part of the research project EMILI (`http://www.emili-project.eu/`) funded by the European Commission under grant agreement number 242438. Special thanks go to Fabian Groffen and Martin Kersten from the MonetDB team at CWI for their support in the prototype implementation of TSA and to Steffen Hausmann for the excellent cooperation in the design of Dura and TSA and for numerous fruitful discussions.

10. REFERENCES

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: A new model and architecture for data stream management. VLDB 2003.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. SIGMOD 2008.

[4] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: Semantic foundations and query execution. VLDB 2006.

[5] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: A stream data management benchmark. VLDB 2004.

[6] R. S. Barga and H. Caituiro-Monge. Event correlation and pattern detection in CEDR. In *Proc. Int. Workshop Reactivity on the Web*. Springer, 2006.

[7] F. Bry and M. Eckert. On static determination of temporal relevance for incremental evaluation of complex event queries. DEBS 2008.

[31] This means a current event of some event type may contribute to future events of the same event type, but neither to present or past events.

[8] B. Chandramouli, J. Goldstein, and D. Maier. High-performance dynamic pattern matching over disordered streams. VLDB 2010.

[9] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[10] E. F. Codd. Relational completeness of data base sublanguages. *In: Database Systems, Prentice Hall*, 1972.

[11] C. Cranor, T. Johnson, and O. Spataschek. Gigascope: a stream database for network applications. SIGMOD 2003.

[12] M. Eckert. *Complex Event Processing with XChange$^{\mathrm{EQ}}$: Language Design, Formal Semantics and Incremental Evaluation for Querying Events*. PhD thesis, Institute for Informatics, University of Munich, 2008.

[13] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann. Two Semantics for CEP, no Double Talk: Complex Event Relational Algebra (CERA) and its Application to XChangeÊQ. In *Reasoning in Event-based Distributed Systems*. Springer, 2011.

[14] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[15] H. Garcia-Molina, J. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall, 2001.

[16] T. M. Ghanem, W. G. Aref, and A. K. Elmagarmid. Exploiting predicate-window semantics over data streams. SIGMOD 2006.

[17] M. Hong. *Expressive and Scaleable Event Stream Processing*. PhD thesis, Cornell University, 2009.

[18] M. K. Kostas Stergiou. Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, 2000.

[19] J. Krämer and B. Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. Database Syst.*, 34(1), April 2009.

[20] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. SIGMOD 2005.

[21] J. Li, K. Tufte, V. Shkapenyuk, V. Papadimos, T. Johnson, D. Maier. Out-of-order processing: a new architecture for high-performance stream systems. VLDB 2008.

[22] E. Liarou, R. Goncalves, and S. Idreos. Exploiting the power of relational databases for efficient stream processing. EDBT 2009.

[23] D. Maier, M. Grossniklaus, S. Moorthy, and K. Tufte. Capturing episodes: may the frame be with you. DEBS 2012.

[24] J. P. Rina Dechter, Itay Meiri. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, May 1991.

[25] P. A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Trans. on Knowl. and Data Eng.*, 2003.

[26] S. Brodt, S. Hausmann, and F. Bry. D4.6: Modularization Mechanisms. `www.emili-project.eu`, 2012.

[27] S. Brodt, S. Hausmann, and F. Bry. D4.7: Refinement of the implementation of event processing and ECA rules for SITE, 2012.

[28] S. Hausmann, S. Brodt, and F. Bry. D4.3: Dura - concepts and examples. `http://www.emili-project.eu/`, 2011.

[29] N. Seifert and M. Bettelini. D3.1: Use Cases Requirements Analysis and Specification (Main Report). `www.emili-project.eu`, 2010.

[30] N. Seifert, M. Bettelini, and S. Rigert. D3.2: Concrete Use Case Models. `www.emili-project.eu`, 2011.

[31] N. Seifert, et al. D3.3: Use case modelling for implementation in SITE. `www.emili-project.eu`, 2012.