Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Efficient Temporal Datalog Materialisation for Composite Event Recognition

Periklis Mantenoglou
Örebro University, Sweden
periklis.mantenoglou@oru.se

Abstract

Several applications demand the timely detection of critical situations, such as threats to safety and transparency, over high-velocity streams of symbolic events. This demand has motivated the development of (i) *event specification languages*, which define composite events via temporal patterns over simpler events, and (ii) *stream reasoning frameworks*, evaluating patterns expressed in these languages. However, event specification languages are typically studied in isolation, complicating their comparison in terms of expressivity and obscuring the scope of their associated stream reasoners. To mitigate this issue, we map practical fragments of prominent event specification languages into Temporal Datalog$^{\rightarrow}_{\ominus}$, a temporal Datalog with stratified negation and no future dependencies. To support efficient stream reasoning over Temporal Datalog$^{\rightarrow}_{\ominus}$, we propose Streaming Trigger Graphs, an extension of a state-of-the-art technique for Datalog materialisation. Our approach yields a uniform composite event recognition mechanism that has the potential to generalise across a wide range of practical event specification languages.

1 Introduction

Composite event recognition (CER) frameworks detect instances of situations of interest, such as safety and transparency threats, over streams of symbolic events (Giatrakos et al. 2020). Effective CER is critical in various domains, such as smart cities (Khazael et al. 2023), health systems (Falcionelli et al. 2019), and fleet management (Tsilionis et al. 2019). A CER solution typically involves two components: an event specification language (ESL), defining a class of composite event patterns via syntactic constructs and operators, and a reasoner evaluating these patterns with minimal latency over unbounded event streams. This separation enables a principled study of the trade-offs between ESL expressivity and reasoning tractability, avoiding ad-hoc solutions, and is thus prominent in CER (Das et al. 2018; Bucchi et al. 2022; Tsilionis et al. 2024; Alevizos et al. 2024; Wang et al. 2025; Mantenoglou et al. 2025).

Although there are various formal studies on specific ESLs (Zhang et al. 2014; Grez et al. 2020; Zielinski 2023; Mantenoglou and Artikis 2025), a unifying approach is lacking, complicating ESL comparisons and obscuring the scope of the associated stream reasoners. This also leads to practical issues as composite event patterns expressed in one ESL often cannot be translated into another ESL in a faithful and optimised manner, which may prohibit the use of the reasoner that is best suited to a specific CER problem while also complicating comparisons between reasoners.

Towards addressing this issue, we compile two prominent logic-based ESLs—based on the Event Calculus (Kowalski and Sergot 1986) and LARS (Beck et al. 2018)—into a variant of Temporal Datalog (Chomicki and Imielinski 1988) with stratified negation and no future dependencies, which we term Temporal Datalog$^{\rightarrow}_{\ominus}$. The Event Calculus has been used in ESLs for large-scale CER (Montali et al. 2013; Artikis et al. 2015; Falcionelli et al. 2019; Baumgartner 2021; Mantenoglou et al. 2023; Tsilionis et al. 2024; Mantenoglou et al. 2025), while LARS features practical fragments for CER (Beck et al. 2017; Bazoobandi et al. 2017), along with extensions for increased expressivity (Eiter and Kiesel 2020; Urbani et al. 2022), and distribution (Eiter et al. 2019).

Temporal Datalog has been studied extensively in the context of stream reasoning (Chomicki and Imielinski 1988; Dantsin et al. 2001; Ronca et al. 2018b; Ronca et al. 2018a; Ronca et al. 2022), while being closely related to expressive ESLs with dedicated stream reasoners, such as DatalogMTL (Walega et al. 2019; Walega et al. 2023; Wang et al. 2025). Thus, Temporal Datalog$^{\rightarrow}_{\ominus}$ serves as a natural formalism for embedding and studying ESLs. To support efficient CER over such embedded specifications, we propose Streaming Trigger Graphs (STGs), an extension of a state-of-the-art materialisation method for Datalog (Tsamoura et al. 2021) that supports Temporal Datalog$^{\rightarrow}_{\ominus}$ programs. An STG leverages a so-called *incremental stratification* of the program to enable delta model updates over streaming data and optimisations that reduce redundant computations.

Our **contributions** may be summarised as follows. **First**, we provide mappings from practical Event Calculus and LARS fragments to Temporal Datalog$^{\rightarrow}_{\ominus}$, and prove that they are equivalence-preserving with respect to query answering. **Second**, we propose STG-based materialisation as a novel reasoning algorithm for Temporal Datalog$^{\rightarrow}_{\ominus}$, demonstrate its correctness and discuss reasoning optimisations. Through the use of Temporal Datalog$^{\rightarrow}_{\ominus}$ and STGs, we provide a uniform CER solution for two prominent ESLs, facilitating their formal analysis and comparison. Our contributions apply to ESLs with stratified negation and no future dependencies, commonly required in stream reasoning (Zaniolo 2012), helping to further consolidate and advance CER.

2 Background

2.1 Logic Programming

We investigate ESLs that are based on logic programming. Logic programs include variables, constants, predicates, and functors. Variables start with an upper-case letter, whereas predicates, functors and constants start with a lower-case letter. A term is either a variable, a constant, or an expression $f(\boldsymbol{X})$, where f is a functor and $\boldsymbol{X}$ is a term tuple. An expression $p(\boldsymbol{X})$, where p is a predicate and $\boldsymbol{X}$ is a term tuple, is called an atom. A fact is a variable-free atom. A literal is an atom optionally preceded by "not", expressing negation-by-failure (Clark 1977). Arity means number of arguments.

A logic program is a set of rules $h \leftarrow b_1, \ldots, b_n$, where $\leftarrow$ denotes implication, atom h is the head of the rule and its body $b_1, \ldots, b_n$ is a conjunction of conditions. All variables in a rule are implicitly universally quantified. In definite (resp. normal) logic programs, all body conditions are atoms (literals). An expression is called ground if it contains no variables. A substitution is a mapping of the terms in an expression to constants. A predicate is called intensional if it appears in the head of a rule with non-empty body, and extensional otherwise. A database D is a set of facts with extensional predicates. A Datalog program is a definite logic program without functors. Every Datalog program can be rewritten into an equivalent program partitioned into extensional rules (with only extensional body predicates) and intensional rules (with only intensional body predicates).

An interpretation of a logic program P is a set of facts from P. To define the semantics of P, we identify its models, i.e., interpretations satisfying all rules in P. Definite logic programs have a unique minimal model, while normal logic programs (which include negation) may have multiple minimal models, complicating their semantic characterisation. To interpret such programs, the closed world assumption (CWA) allows us to infer "not a" if a cannot be finitely proven (Reiter 1977). Thus, for a program with rule "$a \leftarrow b, \mathsf{not}\ c$" and fact b, model $\{b, a\}$ is preferred over model $\{b, c\}$, because, since it is impossible to prove c, the CWA infers "not c", leading to the derivation of a via the rule. When the program does not include cyclic dependencies via not, i.e., it is *stratified*, we can derive an intended interpretation for it by applying the CWA to negated atoms following rule dependencies bottom-up (Przymusinski 1988).

Definition 1 (Global Stratification and Local Stratification)**.** *Consider a normal logic program P with the predicates in set Σ and the ground atoms in set B. A rule r in P is:*

$$h(\boldsymbol{X_h}) \leftarrow b_1(\boldsymbol{X_{b_1}}), \ldots, b_m(\boldsymbol{X_{b_m}}), \quad \mathsf{not}\ c_1(\boldsymbol{X_{c_1}}), \ldots, \mathsf{not}\ c_k(\boldsymbol{X_{c_k}}). \tag{1}$$

P is globally stratified if Σ can be decomposed into disjoint sets (strata) $\Sigma_1, \ldots, \Sigma_n$, so that, for every r in P, we have that, if $h \in \Sigma_i$, where $i \in [n]$, then: (i) $b_t \in \Sigma_j$, where $t \in [m]$ and $j \leq i$, and (ii) $c_t \in \Sigma_j$, where $t \in [k]$ and $j < i$. Such a decomposition is a global stratification of P.

P is locally stratified if B can be split into (possibly infinite) strata $B_1, B_2, \ldots$, so that, for every ground instance of a rule r in P, if $h(\boldsymbol{x_h}) \in B_i$ then: (i) $b_t(\boldsymbol{x_{b_t}}) \in B_j$, where $t \in [m]$ and $j \leq i$, and (ii) $c_t(\boldsymbol{x_{c_t}}) \in B_j$, where $t \in [k]$ and $j < i$. Such a decomposition is a local stratification of P.

Based on Definition 1, every globally stratified program is also locally stratified. Local stratification, however, is undecidable in the general case, while global stratification can be decided in polynomial time (Palopoli 1992). A stratified program admits a unique *perfect model* (Przymusinski 1988), derived by (i) computing a stratification, and (ii) evaluating its rules in a bottom-up order based on the stratum of their head while applying the CWA to negative conditions.

2.2 Temporal Datalog

A temporal program is a logic program where constants are split into objects and time-points, variables are split into object variables and time variables, and time-points are positive integers. An object term is an object or an object variable. A temporal fact is a fact including exactly one time-point. The model of a temporal program is evaluated over a stream, i.e., an unbounded database of temporal facts.

Datalog programs have been studied extensively (Ullman 1989; Shkapsky et al. 2016), and several temporal extensions have been proposed (Zaniolo 2012; Walega et al. 2023; Bellomarini et al. 2025). Such an extension is Datalog$_{1S}$ where each predicate is augmented with an additional argument featuring a *time term* (Chomicki and Imielinski 1988). A time term is T - k, where T is a time variable and k is an integer called the time offset. Integer addition is silently applied when T is grounded. We focus on Temporal Datalog, i.e., a fragment of Datalog$_{1S}$ where each rule may include at most one time variable (Ronca et al. 2022).

To formalise Temporal Datalog$^{\rightarrow}_{\ominus}$, i.e., the Temporal Datalog variant we use for embedding ESLs, we make two modifications on Temporal Datalog: (i) we introduce a restricted form of negation, and (ii) we disallow derivations towards the past. A Temporal Datalog$^{\rightarrow}_{\ominus}$ rule has the following form:

$$\begin{aligned} &h(\boldsymbol{X_h}, T) \leftarrow \\ &\quad b_1(\boldsymbol{X_{b_1}}, T \text{ - } k_{b_1}), \ldots, b_m(\boldsymbol{X_{b_m}}, T \text{ - } k_{b_m}), \\ &\quad \mathsf{not}\ c_1(\boldsymbol{X_{c_1}}, T \text{ - } k_{c_1}), \ldots, \mathsf{not}\ c_k(\boldsymbol{X_{c_k}}, T \text{ - } k_{c_k}). \end{aligned} \tag{2}$$

We restrict negation in Temporal Datalog$^{\rightarrow}_{\ominus}$ by requiring programs to be temporally stratified (Nomikos et al. 2005).

Definition 2 (Temporal Stratification)**.** *Consider a program P with rules* (2)*, and the program P' produced by removing all object terms from P and reducing predicate arity to 1. P is called temporally stratified if P' is locally stratified.*

For a local stratification $B'_1, B'_2, \ldots$ of P', the sequence $B_1, B_2, \ldots$, where $(p, t) \in B_i$ iff $p(t) \in B'_i$, is a temporal stratification of P.

According to Definition 2, if, in a temporally stratified program P, $p(\boldsymbol{X}, T)$ depends on $p(\boldsymbol{X'}, T')$ through negation, then $T' \neq T$. Thus, such dependencies are not cyclic in the grounded program, and can be resolved by unfolding (i.e., grounding) time. As a result, all temporally stratified programs are locally stratified, and thus have a unique perfect model. Temporal stratification can be decided in polynomial time, while local stratification is co-NP hard on Temporal Datalog with negation (Nomikos et al. 2005).

To formalise our second modification on Temporal Datalog, we define a forward-propagating program.

Definition 3 (Forward-Propagating Program)**.** *A program P with rules of the form* (2) *is forward-propagating if, for every time term T - k in the body of a rule in P, k is non-negative.*

Definition 4 (Temporal Datalog$^{\rightarrow}_{\ominus}$ Program)**.** *A Temporal Datalog$^{\rightarrow}_{\ominus}$ program P is a set of forward-propagating rules of the form* (2)*, such that P is temporally stratified.*

Example 1 (Temporal Datalog$^{\rightarrow}_{\ominus}$ Program)**.** *The following program P evaluates the safety status of a device X over a stream of $repair$ and $warning$ temporal facts.*

$$safe(X, T) \leftarrow repair(X, T\text{ - }1). \tag{3}$$

$$safe(X, T) \leftarrow safe(X, T\text{ - }1), \mathsf{not}\ warning(X, T\text{ - }1). \tag{4}$$

$$trusted(X, T) \leftarrow safe(X, T), safe(X, T\text{ - }1), safe(X, T\text{ - }2). \tag{5}$$

Rules (3)–(4) *state that a device X is considered $safe$ at the current time-point T if there was a $repair$ fact with time-stamp T - 1 or it was proven to be $safe$ at T - 1 and no $warning$ was produced at T - 1. Rule* (5) *expresses that a device X is considered $trusted$ if it was proven to be $safe$ for three consecutive time-points (including T). P is forward-propagating and $B_t = \{(p, t) | p \in Pred(P)\}$ defines a temporal stratification. Thus, P is a Temporal Datalog$^{\rightarrow}_{\ominus}$ program.*

2.3 Trigger Graphs

Trigger Graphs (TGs) were proposed as a technique to steer Datalog materialisation away from redundant computations (Tsamoura et al. 2021), complementing classical semi-naive evaluation and the chase-based approaches (Nenov et al. 2015; Urbani et al. 2016; Bellomarini et al. 2018). Towards defining TGs, we start with Execution Graphs (EGs).

Definition 5 (Execution Graph)**.** *An Execution Graph (EG) for a Datalog program P is an acyclic, node- and edge-labeled digraph $G = (V, E, \mathrm{rule}, l)$, where V and E are the set of nodes and the set of edges of the graph, while* rule *and l are the node- and edge-labeling functions.* rule *maps each node in V to a rule in P. There can be a labeled edge of the form $u \rightarrow_j v$, i.e., $(u, v) \in E$ and $l((u, v)) = j$, only if the j-th predicate in the body of* $\mathrm{rule}(v)$ *equals the head predicate of* $\mathrm{rule}(u)$*. For each non-leaf node v, the EG includes exactly one edge $u \rightarrow_j v$ for each body atom of* $\mathrm{rule}(v)$.

Each node v of an EG denotes a plan for executing $\mathrm{rule}(v)$ by specifying the node u that will be used for the evaluation of its i-th body condition through an edge $u \rightarrow_i v$. To enable plan execution, each node v is populated with facts produced by applying $\mathrm{rule}(v)$ on the facts in the nodes specified by its incoming edges.

Definition 6 (Facts in EG)**.** *Let P be a Datalog program, D a database, G an EG for P, v a node in G and σ a substitution for the variables in* $\mathrm{rule}(v)$*. The rules in P are partitioned into extensional and intensional, while h and b denote the head and the body of* $\mathrm{rule}(v)$*. $v(D)$, i.e., the set of facts in node v given D, contains a fact $h\sigma$ if:*

- $\mathrm{rule}(v)$ *is an extensional rule and for each atom b_i in b we have $b_i\sigma \in D$, or*
- $\mathrm{rule}(v)$ *is an intensional rule and for each atom b_i in b we have $b_i\sigma \in u_i(D)$, where $u \rightarrow_i v$ is an edge in G.*

Algorithm 1 TGmat

Input: A Datalog program P and a database D.
Output: The facts in a materialised TG for $P \cup D$.

1: $k := 0$, $G := \emptyset$, $I^0 := \emptyset$
2: **repeat**
3: $\quad k := k + 1$, $I^k := I^{k-1}$
4: $\quad$ **for all** rules r in P **do**
5: $\quad\quad$ **if** r is extensional **and** $\exists \sigma : \forall b_i \in r : b_i\sigma \in D$ **then**
6: $\quad\quad\quad G$.add$((v, \emptyset, \mathrm{rule}(v) = r, \emptyset))$
7: $\quad\quad$ **else**
8: $\quad\quad\quad$ **for all** $(u_1, \ldots, u_n)$ k-compatible with r **do**
9: $\quad\quad\quad\quad G$.add$((v, \bigcup_{i\in[n]}(u_i, v), \mathrm{rule}(v) = r, \bigcup_{i\in[n]} l((u_i, v)) = i))$
10: $\quad G := \textsc{minDatalog}(G)$
11: $\quad$ **for all** nodes v in G of depth k **do** I^k.add$(v(D, I^{k-1}))$
12: **until** $I^k = I^{k-1}$
13: **return** I^k

$G(D) = D \cup \bigcup_{v \in V} v(D)$ contains all facts derived by materialising EG G on database D.

If $G(D)$ contains all facts that follow from $P \cup D$, and no other facts, then we say that EG G is a TG.

Definition 7 (Trigger Graph)**.** *Considering a Datalog program P, a database D, and an EG G for P, we say that G is a Trigger Graph (TG) for $P \cup D$ if for each fact a we have $P \cup D \models a$ iff $a \in G(D)$. G is a TG for P if, for each database D, G is a TG for $P \cup D$.*

k-compatibility determines which nodes of an EG contain facts that may be combined to derive new facts.

Definition 8 (k-compatibility)**.** *Let P be a program, r an intensional rule in P and G an EG for P. A tuple $(u_1, \ldots, u_n)$ of nodes from G is k-compatible with r if:*

- *the predicate in the head of u_i is equal to the predicate in the i-th body atom of r;*
- *the depth of each u_i, i.e., the length of the longest path in G that ends at u_i, is less than k; and*
- *at least one node in $(u_1, \ldots, u_n)$ has depth k - 1.*

Algorithm 1 outlines the process of building a TG G for program P and database D. In the first iteration ($k = 1$), we add one node in G for each extensional rule that fires based on D (lines 5–6). In all subsequent iterations ($k > 1$), we use k-compatibility to determine whether a node v for rule r and incoming edges from nodes $u_1, \ldots, u_n$ will be constructed (lines 8–9). The third requirement in Definition 8 ensures that node combinations used in previous iterations are not k-compatible, thereby preventing redundant node constructions, in line with semi-naïve evaluation (Ullman 1989). After each iteration, we remove the nodes of depth k that are subsumed by another node with respect to the query they express (line 10), and then populate set I^k with the facts in the remaining nodes of depth k (line 11). $v(D, I^{k-1})$ denotes an optimised materialisation strategy for these facts, restricting attention to tuples that have not been derived at a previous step, i.e., they are not in I^{k-1}. We terminate if no new facts were added in the last iteration (line 13).

3 Event Specification Languages

We focus on two prominent logical ESLs based on LARS and the Event Calculus. A thorough review of ESLs with informative comparisons is provided in Section 6.

3.1 LARS

LARS extends ASP with window operators and temporal modalities for stream reasoning (Beck et al. 2018). A window operator $\boxplus^w a$ over formula a uses window function w to restrict its evaluation on a particular subset of a stream S. For example, time windows $\boxplus^n$ restrict attention to the events that took place over the last n time-points. Temporal modalities express the temporal validity of a formula a. $\Diamond\, a$ (resp. $\Box\, a$) requires that a holds at some (every) time-point in S. $@_{T'} a$ requires that a holds when the evaluation time is shifted to T', granted that $T' \in S$. In practice, it is used to express that a holds at some time-point in S (i.e., same as $\Diamond$), but also to bind T' to such a time-point.

The main LARS-based stream reasoners are Ticker (Beck et al. 2017) and Laser (Bazoobandi et al. 2017). Both are based on the "plain LARS" fragment, whose core building blocks are the so-called *extended atoms*.

Definition 9 (Extended atom)**.** *Let a be an atom $p(\boldsymbol{X})$, where p is a predicate and $\boldsymbol{X}$ is a sequence of object terms, T' a time variable and d a positive integer. An extended atom e is defined by the following grammar:*

$$e ::= a \mid @_{T'} a \mid \boxplus^d @_{T'} a \mid \boxplus^d \Diamond\, a \mid \boxplus^d \Box\, a$$

We focus on time windows and rules without $@_{T'}$ operators in the head, leading to forward-propagating programs.

Definition 10 (Plain LARS)**.** *A plain LARS program is a set of rules of the form "$h \leftarrow b_1, \ldots, b_m$.", where h is an atom, b_i, $i \in [m]$, is a (possibly negated) extended atom, and all window operators are time windows.*

In plain LARS, if a rule r includes condition $@_{T'} a$, then T' must be grounded earlier in the body of r, in order to avoid evaluating $@_{T'} a$ over arbitrary future time-points. Therefore, every such condition is preceded by a positive $\boxplus^d @_{T'} a$ condition, which constrains T' to the window $\{T\text{-}\,d, \ldots, T\}$, where T is the rule evaluation time.

While both Ticker and Laser employ plain LARS, they differ on their support for negation. Ticker allows non-stratified negation, while Laser is restricted to globally stratified programs. We focus on globally stratified plain LARS programs, motivated by the superior reasoning efficiency of Laser (Beck et al. 2018), as well as by the common assumption in CER that there is a single, implicit model of the world expressing the actual composite event occurrences (Cugola and Margara 2012).

Example 2 (Plain LARS Program)**.** *The following plain LARS program monitors safety in a network of devices.*

$$verified(X) \leftarrow \boxplus^2 \Diamond\, repair(X). \tag{6}$$

$$trusted(X) \leftarrow \boxplus^6 \Box\, verified(X). \tag{7}$$

$$unverified(X) \leftarrow \boxplus^2 \Box \neg repair(X). \tag{8}$$

$$\begin{aligned} integrity\_threat\_of(X, Y) \leftarrow \\ \boxplus^2 \Box\, trusted(X), \boxplus^2 @_{T'} unverified(Y), \\ \boxplus^2 @_{T'} connected(X, Y). \end{aligned} \tag{9}$$

Rules (6)–(7) *state that device X is verified if a repair occurred in a two time-step window, and trusted if it has been verified over a six time-step window. Rule* (8) *marks X as unverified if no repair occurred in a two time-step window, while rule* (9) *reports that a device Y poses an integrity threat to a trusted device X if Y was unverified and connected to X at some time-point T' within a two time-step window. A global stratification places repair in the first stratum and the remaining predicates in the second stratum.*

3.2 Event Calculus with Delayed Effects

The Event Calculus (EC) is a temporal logic programming formalism where the functors are classified as event types and *fluent* types, expressing time-varying properties, and a constant sort is reserved for the possible values of each fluent type (Kowalski and Sergot 1986). An event $e(\boldsymbol{X})$ (resp. fluent $f(\boldsymbol{X})$) is a term where e (f) is an event (fluent) type and $\boldsymbol{X}$ contains only objects and variables. Expression $f(\boldsymbol{X}) = v$ states that fluent $f(\boldsymbol{X})$ has value v and is often used in CER to denote composite events (Artikis et al. 2015; Falcionelli et al. 2019). We focus on an EC variant where events may have **d**elayed **e**ffects on fluent values—which we term EC$^{\text{DE}}$—as it is employed in a state-of-the-art CER framework (Mantenoglou et al. 2025). The predicates in EC$^{\text{DE}}$ and their meaning are as follows.

- happensAt$(e(\boldsymbol{X}), T)$: event $e(\boldsymbol{X})$ occurs at time T.
- holdsAt$(f(\boldsymbol{X}) = v, T)$: fluent $f(\boldsymbol{X})$ has value v at T.
- initiatedAt$(f(\boldsymbol{X}) = v, T)$/terminatedAt$(f(\boldsymbol{X}) = v, T)$: a time period where $f(\boldsymbol{X}) = v$ is initiated/terminated at T.
- fi$(f(\boldsymbol{X}) = v, f(\boldsymbol{X}) = v', d)$: an initiation of $f(\boldsymbol{X}) = v$ leads to a **f**uture **i**nitiation of $f(\boldsymbol{X}) = v'$ after d time-points, unless $f(\boldsymbol{X}) = v$ is terminated in the meantime.
- p$(f(\boldsymbol{X}) = v)$: the above future initiation of $f(\boldsymbol{X}) = v'$ is **p**ostponed by intermediate initiations of $f(\boldsymbol{X}) = v$.

In EC$^{\text{DE}}$, fluents are classified as *simple* or *statically determined*. The values of a simple fluent change based on the satisfaction of initiation and termination conditions and persist via the law of inertia. A statically determined fluent is defined as a Boolean combination of other fluents.

Simple fluent definitions. The rules defining $f(\boldsymbol{X_f}) = v$ based on immediate effects of events have this form:

$$\begin{aligned} &\text{initiatedAt}(f(\boldsymbol{X_f}) = v,\ T) \leftarrow \\ &\quad \text{happensAt}(e_1(\boldsymbol{X_{e_1}}),\ T)[, \\ &\quad [\text{not}]\ \text{happensAt}(e_2(\boldsymbol{X_{e_2}}),\ T),\ \ldots \\ &\quad [\text{not}]\ \text{happensAt}(e_n(\boldsymbol{X_{e_n}}),\ T), \\ &\quad [\text{not}]\ \text{holdsAt}(f_1(\boldsymbol{X_{f_1}}) = v_1,\ T),\ \ldots, \\ &\quad [\text{not}]\ \text{holdsAt}(f_m(\boldsymbol{X_{f_m}}) = v_m,\ T)]. \end{aligned} \tag{10}$$

We use "[ ]" to denote optional parts of the rule, i.e., only the first positive happensAt atom is necessary. Rules with head terminatedAt$(f(\boldsymbol{X_f}) = v, T)$ have the same form.

fi and p facts express future initiations as follows:

$$\begin{aligned} &\text{initiatedAt}(F = V', T{+}d) \leftarrow \\ &\quad \text{fi}(F = V, F = V', d), \text{initiatedAt}(F = V, T), \\ &\quad \text{not cancelled}(F = V, T, T{+}d). \end{aligned} \tag{11}$$

$$\begin{aligned} &\text{cancelled}(F = V, T, T + d) \leftarrow \\ &\quad \text{termIn}(F = V, T, T + d). \end{aligned} \tag{12}$$

$$\begin{aligned} &\text{cancelled}(F = V, T, T + d) \leftarrow \\ &\quad \text{p}(F = V), \text{initIn}(F = V, T, T + d). \end{aligned} \tag{13}$$

$\mathsf{initIn}(F = V, T, T + d)$ and $\mathsf{termIn}(F = V, T, T + d)$ express that $F = V$ is initiated or terminated, respectively, at some time-point strictly between T and $T + d$. In rules (11)–(13), F is a variable spanning over fluents, while V and V' span over its possible values. This abstraction emphasises that these rules are included regardless of the domain being modelled. However, they may only fire if fi—and possibly p—facts are included, expressing the dynamics of delayed fluent value changes in the domain.

EC$^{\text{DE}}$ also uses the following rules regardless of domain.

$$\mathsf{holdsAt}(F = V, T) \leftarrow \mathsf{initiatedAt}(F = V, T\text{ - }1). \quad (14)$$

$$\mathsf{holdsAt}(F = V, T) \leftarrow \mathsf{holdsAt}(F = V, T\text{ - }1), \ \mathsf{not\ terminatedAt}(F = V, T\text{ - }1). \quad (15)$$

$$\mathsf{terminatedAt}(F = V, T) \leftarrow \mathsf{initiatedAt}(F = V', T), V' \neq V. \quad (16)$$

Rules (14)–(15) express the law of inertia, i.e., $F = V$ holds at T if it was initiated at the previous time-point T - *1* or held at T - *1* and it was not terminated at T - *1*. Rule (16) terminates $F = V$ when F is initiated with a value other than V, restricting fluents to at most one value at a time.

Statically determined fluent definitions. Such a definition for $f(\boldsymbol{X_f}) = v$ contains m rules where the i-th rule is:

$$\mathsf{holdsAt}(f(\boldsymbol{X_f}) = v,\ T) \leftarrow [\mathsf{not}]\ \mathsf{holdsAt}(f_{i1}(\boldsymbol{X_{f_{i1}}}) = v_{i1},\ T)[, [\mathsf{not}]\ \mathsf{holdsAt}(f_{i2}(\boldsymbol{X_{f_{i2}}}) = v_{i2},\ T), \ldots, [\mathsf{not}]\ \mathsf{holdsAt}(f_{in}(\boldsymbol{X_{f_{in}}}) = v_{in},\ T)]. \quad (17)$$

Together, these m rules define $f(\boldsymbol{X_f}) = v$ as a DNF formula on the values held by the fluents in their bodies. This definition has proven to be equivalent to the interval-based analog used for practical CER (Mantenoglou and Artikis 2025).

EC$^{\text{DE}}$ programs are locally stratified, and thus admit a unique perfect model (Mantenoglou et al. 2025). The reasoning task in EC$^{\text{DE}}$ is to compute holdsAt, i.e., the values of fluents at each time-point.

Example 3 (EC$^{\text{DE}}$ Program)**.** *An EC^{DE} program monitoring device safety may include the following rules.*

$$\mathsf{initiatedAt}(\mathit{safety}(X) = \mathit{verified}, T) \leftarrow \mathsf{happensAt}(\mathit{repair}(X), T). \quad (18)$$

$$\mathsf{initiatedAt}(\mathit{safety}(X) = \mathit{unverified}, T) \leftarrow \mathsf{happensAt}(\mathit{warning}(X), T). \quad (19)$$

$$\mathsf{holdsAt}(\mathit{integrity\_threat\_of}(X, Y) = \mathsf{true}, T) \leftarrow \mathsf{holdsAt}(\mathit{safety}(X) = \mathit{trusted}, T), \mathsf{holdsAt}(\mathit{safety}(Y) = \mathit{unverified}, T), \mathsf{holdsAt}(\mathit{connected}(X, Y) = \mathsf{true}, T). \quad (20)$$

$$\mathsf{fi}(\mathit{safety}(X) = \mathit{verified}, \mathit{safety}(X) = \mathit{trusted}, 3). \quad (21)$$

Rule (18) *(resp. rule* (19)*) states that a device X is considered verified (unverified) after a repair(X) (warning(X)) event occurs. Rule* (20) *expresses that an unverified device Y poses an integrity threat to a trusted device X if they are connected. Fact* (21) *expresses that X becomes trusted after being verified for three consecutive time-points.*

Algorithm 2 pLARS2TD

Input: A plain LARS program P.
Output: A Temporal Datalog$^{\rightarrow}_{\ominus}$ program P'.

1: $Pred(P') := \emptyset$, $P' := \emptyset$
2: **for all** predicates p/n in P **do** $Pred(P').\mathsf{add}(p/(n+1))$
3: $all\_rules\_t := \emptyset$
4: **for all** rules $r \in P$ **do**
5: $\quad rules\_t := \{(r, \emptyset)\}$
6: $\quad$ **for all** time variables T' appearing in r **do**
7: $\quad\quad$ **let** $\boxplus^d @_{T'} p(\boldsymbol{X})$ **be** $first\_positive\_at(r, T')$
8: $\quad\quad$ **for all** $(r', \mu) \in rules\_t$ **do**
9: $\quad\quad\quad rules\_t.\mathsf{remove}((r', \mu))$
10: $\quad\quad\quad$ **for all** $i{:}0{\leq}i{\leq}d$ **do** $rules\_t.\mathsf{add}((r', \mu \cup \{T' \mapsto T\text{ - }i\}))$
11: $\quad\quad all\_rules\_t.\mathsf{add\_all}(rules\_t)$
12: **for all** $(r, \mu) \in all\_rules\_t$ **do**
13: $\quad replace(r, head(r), p_h(\boldsymbol{X}, T))$ **where** $head(r) = p_h(\boldsymbol{X})$
14: $\quad$ **for all** body conditions $b \in r$ **do**
15: $\quad\quad$ **if** b **is** $p(\boldsymbol{X})$ **then** $replace(r, b, p(\boldsymbol{X}, T))$
16: $\quad\quad$ **else if** b **is** $@_{T'} p(\boldsymbol{X})$ **then** $replace(r, b, p(\boldsymbol{X}, \mu(T')))$
17: $\quad\quad$ **else if** b **is** $\boxplus^d @_{T'} p(\boldsymbol{X})$ **then**
18: $\quad\quad\quad$ **if** $T\text{ - }d \leq \mu(T')$ **then** $replace(r, b, p(\boldsymbol{X}, \mu(T')))$
19: $\quad\quad\quad$ **else** $replace(r, b, \bot)$
20: $\quad\quad$ **else if** b **is** $\boxplus^d \Diamond p(\boldsymbol{X})$ **then**
21: $\quad\quad\quad Pred(P').\mathsf{add}(p^d_{\Diamond})$
22: $\quad\quad\quad$ **for all** $i{:}0{\leq}i{\leq}d$ **do** $P'.\mathsf{add}(p^d_{\Diamond}(\boldsymbol{X}, T){\leftarrow}p(\boldsymbol{X}, T\text{ - }i))$
23: $\quad\quad\quad replace(r, b, p^d_{\Diamond}(\boldsymbol{X}, T))$
24: $\quad\quad$ **else if** b **is** $\boxplus^d \Box p(\boldsymbol{X})$ **then**
25: $\quad\quad\quad Pred(P').\mathsf{add}(p^d_{\Box})$
26: $\quad\quad\quad P'.\mathsf{add}(p^d_{\Box}(\boldsymbol{X}, T){\leftarrow}p(\boldsymbol{X}, T), \ldots, p(\boldsymbol{X}, T\text{ - }d))$
27: $\quad\quad\quad replace(r, b, p^d_{\Box}(\boldsymbol{X}, T))$
28: $\quad P'.\mathsf{add}(r)$
29: **return** P'

4 Embedding ESLs in Temporal Datalog$^{\rightarrow}_{\ominus}$

Plain LARS to Temporal Datalog$^{\rightarrow}_{\ominus}$. A plain LARS program P is not Temporal Datalog$^{\rightarrow}_{\ominus}$ because it includes temporal operators (Definition 10). We propose a translation of P into a Temporal Datalog$^{\rightarrow}_{\ominus}$ program P' that is equivalent with respect to query evaluation for any input stream.

Algorithm 2 outlines our translation. To construct P', we first introduce predicates that denote the atoms in P. For each predicate p with arity n in P, we introduce a predicate p with arity $n + 1$ in P', where the extra argument is reserved for a time term (see line 2). In order for P' to be Temporal Datalog$^{\rightarrow}_{\ominus}$, each rule in P' must contain one time variable T, reflecting its evaluation time (see rule schema (2)). To cater for this, we need to map, in each rule r of P, each time variable T' appearing in $@_{T'}$ operators to time terms T - k. In plain LARS, T' is grounded based on the first positive $\boxplus^d @_{T'} p(\boldsymbol{X})$ body condition of r. Thus, we replace r with $d + 1$ rules such that in the i-th rule, where $i \in \{0, \ldots, d\}$, T' is mapped to T - i, denoted via a mapping μ (see lines 3–11). This reflects the semantics of $@_{T'} p(\boldsymbol{X})$ as an existential quantifier that also maintains via T' the time-points in the window where $p(\boldsymbol{X})$ holds.

Example 4 (Plain LARS to Temporal Datalog$^{\rightarrow}_{\ominus}$)**.** *Consider the plain LARS program in Example 2. Rule* (9) *contains*

one time variable T', grounded via $\boxplus^2 @_{T'} unverified(Y)$. Thus, Algorithm 2 produces three instances of this rule, with mappings $T' \mapsto T$ - 2, $T' \mapsto T$ - 1 and $T' \mapsto T$.

Afterwards, we iterate over each discovered (r, μ) pair and substitute each atom in r with a Temporal Datalog$^{\rightarrow}_{\ominus}$ atom (lines 12–27). For the head $p_h(\boldsymbol{X})$ of r, we append to $\boldsymbol{X}$ a time variable T denoting the rule evaluation time (line 13). Then, we iterate over each body condition b in r. If b contains negation, we implicitly replace "$\neg$" with "not", as their semantics coincide for globally stratified plain LARS programs (Bazoobandi et al. 2017). Then, setting negation aside, we distinguish the following cases. If b is $p(\boldsymbol{X})$ (resp. $@_{T'} p(\boldsymbol{X})$), then we replace b with $p(\boldsymbol{X}, T)$ ($p(\boldsymbol{X}, \mu(T'))$) in r (lines 15–16). If b is $\boxplus^d @_{T'} p(\boldsymbol{X})$, then we distinguish two cases. If $\mu(T')$ falls within $[T\text{ - }d, T]$, then we replace b with $p(\boldsymbol{X}, \mu(T'))$ (line 18). Otherwise, if $\mu(T') < T\text{ - }d$, then we cannot satisfy b, and thus disable the rule by replacing b with $\bot$ (line 19). If b is $\boxplus^d \Diamond p(\boldsymbol{X})$ (resp. $\boxplus^d \Box p(\boldsymbol{X})$), we introduce predicate $p^d_{\Diamond}$ ($p^d_{\Box}$) with arity $n + 1$ and replace b with $p^d_{\Diamond}(\boldsymbol{X}, T)$ ($p^d_{\Box}(\boldsymbol{X}, T)$) (lines 20–27). Then, we add rules in P' that define $p^d_{\Diamond}(\boldsymbol{X}, T)$ ($p^d_{\Box}(\boldsymbol{X}, T)$) as the disjunction (conjunction) of $p(\boldsymbol{X}, T), \ldots, p(\boldsymbol{X}, T\text{ - }d)$.

Example 5 (Example 4 cont'd)**.** *To translate rule* (6)*, Algorithm 2 replaces its head with $verified(X, T)$, its body with $repair^2_{\Diamond}(X, T)$ and introduces three rules of the form: "$repair^2_{\Diamond}(X, T) \leftarrow repair(X, T\text{ - }i)$.", for $i \in \{0, 1, 2\}$. Rules* (7)–(8) *are translated similarly by mapping the $\boxplus^d \Box$ operators they contain according to lines 24–27. Rule* (9) *is translated into three rules, one for each mapping of T' (see Example 4). For $T' \mapsto T$ - 1, e.g., we construct this rule:*

$$integrity\_threat\_of(X, Y, T) \leftarrow trusted^2_{\Box}(X, T), unverified(Y, T\text{ - }1), connected(X, Y, T\text{ - }1).$$

Theorem 1 (pLARS2TD Correctness)**.** *Consider a plain LARS program P, a stream S and let Algorithm 2 map P to P'. P' is Temporal Datalog$^{\rightarrow}_{\ominus}$, and given that I^* is the perfect model of $P' \cup S$, at time-point t, $P \cup S$ derives:*

- $p(\boldsymbol{X})$ *iff* $I^* \models p(\boldsymbol{X}, t)$
- $@_{t'} p(\boldsymbol{X})$ *iff* $I^* \models p(\boldsymbol{X}, t')$
- $\boxplus^d @_{t'} p(\boldsymbol{X})$ *iff* $I^* \models p(\boldsymbol{X}, t')$ *and* $t\text{ - }d \leq t' \leq t$.
- $\boxplus^d \Diamond p(\boldsymbol{X})$ *iff* $I^* \models p^d_{\Diamond}(\boldsymbol{X}, t)$
- $\boxplus^d \Box p(\boldsymbol{X})$ *iff* $I^* \models p^d_{\Box}(\boldsymbol{X}, t)$.

Proof Sketch. Rules in P' each contain one time variable, are forward-propagating (see, e.g., lines 22, 26 and 10), and P' is temporally stratified since P is globally stratified and Algorithm 2 introduces no negative dependencies. Thus, P' is Temporal Datalog$^{\rightarrow}_{\ominus}$ (Definition 4). For model equivalence, we follow an inductive proof on the global stratification $\Sigma_1, \ldots, \Sigma_n$ of P. For extensional predicates, correctness follows from the faithfulness of the translation: e.g., $\boxplus^d \Diamond p(\boldsymbol{X})$ holds iff $p(\boldsymbol{X})$ holds now or at least once over the last d time-points, matching the definition of $p^d_{\Diamond}$ (line 22). For the inductive step at Σ_i, negative body conditions in any rule defining $p \in \Sigma_i$ refer to lower strata and are thus evaluated identically by P and P'. Correctness then reduces to the faithfulness of the translation of the temporal operators, which guarantees that the positive conditions are also evaluated equivalently. □

Algorithm 3 EC2TD

Input: An EC$^{\text{DE}}$ program P.
Output: A Temporal Datalog$^{\rightarrow}_{\ominus}$ program P'.

1: $Pred(P') := \emptyset$, $P' := \emptyset$
2: **for all** event types e/n in P **do** $Pred(P')$.add$(e/(n+1))$
3: **for all** fluent types f/n in P **do**
4: $Pred(P')$.add$(f^h/(n+2))$
5: **if** f simple **then** $Pred(P')$.add_all$(f^i/(n+2), f^t/(n+2))$
6: **for all** rules $r \in P$ following schema (10) or schema (17) **do**
7: P'.add(mapEC$(head(r)) \leftarrow \bigwedge_{b \in body(r)}$[not] mapEC$(b))$
8: **for all** simple fluent types f **and** values v of f **do**
9: P'.add$(f^h(\boldsymbol{X}, v, T) \leftarrow f^i(\boldsymbol{X}, v, T\text{ - }1))$
10: P'.add$(f^h(\boldsymbol{X}, v, T) \leftarrow f^h(\boldsymbol{X}, v, T\text{ - }1),$ not $f^t(\boldsymbol{X}, v, T\text{ - }1))$
11: **for all** values $v' \neq v$ of f **do**
12: P'.add$(f^t(\boldsymbol{X}, v, T) \leftarrow f^i(\boldsymbol{X}, v', T))$
13: **for all** fi$(f(\boldsymbol{X}) = v, f(\boldsymbol{X}) = v', d) \in P$ **do**
14: $b := f^i(\boldsymbol{X}, v, T\text{ - }d),$ not $f^t(\boldsymbol{X}, v, T\text{ - }d+1), \ldots,$ not $f^t(\boldsymbol{X}, v, T\text{ - }1)$
15: **if** p$(f(\boldsymbol{X}) = v) \in P$ **then**
16: b.add(not $f^i(\boldsymbol{X}, v, T\text{ - }d+1), \ldots,$ not $f^i(\boldsymbol{X}, v, T\text{ - }1))$
17: P'.add$(f^i(\boldsymbol{X}, v', T) \leftarrow b)$
18: **return** P'
19: **function** mapEC(a)
20: **case** a: happensAt$(e(\boldsymbol{X}), T)$ **return** $e(\boldsymbol{X}, T)$
21: **case** a: initiatedAt$(f(\boldsymbol{X}) = v, T)$ **return** $f^i(\boldsymbol{X}, v, T)$
22: **case** a: terminatedAt$(f(\boldsymbol{X}) = v, T)$ **return** $f^t(\boldsymbol{X}, v, T)$
23: **case** a: holdsAt$(f(\boldsymbol{X}) = v, T)$ **return** $f^h(\boldsymbol{X}, v, T)$

EC$^{\text{DE}}$ to Temporal Datalog$^{\rightarrow}_{\ominus}$. We consider an EC$^{\text{DE}}$ program P consisting of rules (11)–(16), as well as rules following schemata (10) and (17), and possibly fi and p facts. P contains functors, in the form of event and fluent types, atoms with multiple time terms (see rules (11)–(13)), and an inequality comparison in rule (16). These elements are not in Temporal Datalog$^{\rightarrow}_{\ominus}$ and thus we need to translate P into a Temporal Datalog$^{\rightarrow}_{\ominus}$ program P'.

Algorithm 3 outlines the translation steps. Towards an equivalent program without functors, for each event type e in P, we introduce to P' one predicate e, adding one more argument to express time (see line 2). Similarly, for each fluent type f in P we introduce a predicate f^h in P', denoting when the fluent holds, and equip it with a value argument and a time argument (lines 3–4). If f is a simple fluent type, then we additionally introduce predicates f^i and f^t to denote its initiations and terminations (line 5). As denoted by the mapEC function (lines 19–23), these new predicates correspond to EC$^{\text{DE}}$ predicates for particular events and fluents.

Next, we compile each domain rule, following schema (10) or schema (17), by mapping its head and body atoms to the introduced Temporal Datalog$^{\rightarrow}_{\ominus}$ predicates (lines 6–7). Rules (14)–(16) involve fluent variables, while the predicates we introduced are fluent-specific. To cater for this, we add in P' one such rule for each fluent type f in P, while

grounding variable V according to the possible values of f (lines 8–12). We follow a similar route for rules (11)–(13), while additionally compiling away the predicates with two time terms by explicitly adding a body condition referring to each time-point in the range they define (lines 13–17).

Example 6 (EC$^{\text{DE}}$ to Temporal Datalog$^{\rightarrow}_{\ominus}$)**.** *Consider the EC^{DE} program in Example 3. Algorithm 3 translates rules* (18)–(20) *by mapping their atoms using* mapEC*. Rule* (18)*, e.g., is translated into: "$safety^i(X, verified, T) \leftarrow repair(X, T)$.". Fact* (21) *is expressed in Temporal Datalog$^{\rightarrow}_{\ominus}$ using the following rule:*

$$\begin{aligned} safety^i(X, trusted, T) \leftarrow \\ safety^i(X, verified, T\text{ - }3), \\ \textsf{not } safety^t(X, verified, T\text{ - }2), \\ \textsf{not } safety^t(X, verified, T\text{ - }1). \end{aligned}$$

Moreover, for each value of $safety(X)$, we construct rules expressing its default persistence and its incompatibility with other values. For value $trusted$, e.g., we have:

$$\begin{aligned} safety^h(X, trusted, T) &\leftarrow safety^i(X, trusted, T\text{ - }1). \\ safety^h(X, trusted, T) &\leftarrow safety^h(X, trusted, T\text{ - }1), \\ &\qquad \textsf{not } safety^t(X, trusted, T\text{ - }1). \\ safety^t(X, trusted, T) &\leftarrow safety^i(X, unverified, T). \\ safety^t(X, trusted, T) &\leftarrow safety^i(X, verified, T). \end{aligned}$$

Theorem 2 (EC2TD Correctness)**.** *Consider an EC^{DE} program P, a stream S and let Algorithm 3 map P to P'. P' is Temporal Datalog$^{\rightarrow}_{\ominus}$, and, for each fluent $f(\boldsymbol{X})$, value v of f and time-point t, $P \cup S$ derives* $\textsf{holdsAt}(f(\boldsymbol{X}) = v, t)$ *iff $I^* \models f^h(\boldsymbol{X}, v, t)$, where I^* is the perfect model of $P' \cup S$.*

Proof sketch. All rules added by Algorithm 3 contain one time variable and are forward-propagating. Negation appears (i) on conditions with a positive temporal offset (e.g., line 10), (ii) in a globally stratified statically determined fluent definition, lacking cyclic dependencies (Mantenoglou et al. 2022), and (iii) in an f^h condition of a rule defining f'^i or f'^t ($f \neq f'$). Case (iii) preserves temporal stratification as f^h may depend negatively on f'^i or f'^t only via a positive time offset. Therefore, P' is Temporal Datalog$^{\rightarrow}_{\ominus}$ (Definition 4). Semantic equivalence follows from the faithfulness of the translation up to predicate renaming. □

5 Streaming Trigger Graphs

We cannot use TGs to materialise Temporal Datalog$^{\rightarrow}_{\ominus}$ programs because (i) they do not support negation and (ii) they do not support variables with countably infinite groundings, such as time variables. The presence of such variables makes the order of rule evaluation crucial, as materialisation can become trapped in an infinite chain of derivations for one predicate, thereby preventing the derivation of facts for other predicates. Tackling this issue by generating a TG G^t that materialises the program up to the latest time point t is problematic because incrementally updating G^t in the presence of new data may lead to incorrect results, while recomputing a TG from scratch at every time step is prohibitively expensive in the streaming setting. To address these issues, we propose Streaming Trigger Graphs (STGs), which constitute an optimised computational model for Temporal Datalog$^{\rightarrow}_{\ominus}$.

We start with Streaming Execution Graphs (SEGs), which extend EGs with negation and time. Mirroring the relationship between EGs and TGs, we then define STGs as SEGs that materialise Temporal Datalog$^{\rightarrow}_{\ominus}$ programs correctly.

Definition 11 (Streaming Execution Graph)**.** *A Streaming Execution Graph (SEG) for a Temporal Datalog$^{\rightarrow}_{\ominus}$ program P is a graph $G = (V, E^+, E^-, \text{rule}, \text{time}, l)$, where:*

- *V is a set of nodes.*
- *E^+ is a set of directed edges called p-edges.*
- *E^- is a set of n-edges, i.e., labelled directed hyperedges $(U, \{v\})$, where $U \subset V$.*
- *rule is a function mapping each node in V to a rule in P.*
- *time is a mapping of each node in V to a time-point.*
- *l is a function labelling each p-edge (u, v) in E^+ and each n-edge $(U, \{v\})$ in E^- with a positive integer j, denoted respectively as $u \rightarrow_j v$ and $U \rightarrow_j \{v\}$. We have $u \rightarrow_j v$ (resp. $u \in U$, where $U \rightarrow_j \{v\}$), only if:*
  - *the predicate of the j-th positive (negative) condition of $\text{rule}(v)$ equals the head predicate of $\text{rule}(u)$, and*
  - *$\text{time}(u)$ is equal to $\text{time}(v)$ minus the time offset of the j-th positive (negative) body condition in $\text{rule}(v)$.*

For each non-leaf node v, there is exactly one p-edge $u \rightarrow_j v$ or n-edge $U \rightarrow_j \{v\}$ for the j-th condition in $\text{rule}(v)$.

Considering a rule r with head h, and sets b^+ and b^-, containing respectively the atoms in positive and in negative body conditions of r, if there is a substitution σ on the variables in r such that $b_i^+\sigma$ holds for all b_i^+ in b^+, r derives $h\sigma$ only if $b_i^-\sigma$ does not hold for any b_i^- in b^-. Intuitively, the atoms in b^- act as filters that disqualify substitutions satisfying at least one such atom. To reflect this, Definition 11 uses an n-edge $U \rightarrow_i \{v\}$, where $\text{rule}(v) = r$, to denote that if at least one node $u \in U$ contains fact $b_i^-\sigma$ then fact $h\sigma$ should not be introduced in v. Moreover, according to Definition 11, SEGs restrict body condition evaluation over facts with the appropriate time offset compared to the head predicate. This minimises the atoms considered in joins during materialisation without compromising correctness.

Definition 12 (Facts in SEG)**.** *Let P be a Temporal Datalog$^{\rightarrow}_{\ominus}$ program, S a stream, G a SEG for P, v a node in G and σ a substitution for $\text{rule}(v)$. The rules in P are partitioned into extensional and intensional. h and b^+ (resp. b^-) are the head and the set of atoms in positive (negative) conditions of $\text{rule}(v)$. $v(S)$ contains a fact $h\sigma$ if:*

- *$\text{rule}(v)$ is extensional and for each atom b_i in b^+ (b^-) we have $b_i\sigma \in S$ ($b_i\sigma \notin S$), or*
- *$\text{rule}(v)$ is intensional, for each atom b_i in b^+ we have $b_i\sigma \in u(S)$, where $u \rightarrow_i v$ is a p-edge in G, and for each atom b_i in b^- and each node $u \in U$, where $U \rightarrow_i \{v\}$ is an n-edge in G, we have $b_i\sigma \notin u(S)$.*

$G(S) = S \cup \bigcup_{v \in V} v(S)$ contains all facts derived by materialising SEG G on stream S.

If $G(S)$ contains all facts that follow from $P \cup S$, and no other facts, then we say that SEG G is an STG.

Definition 13 (Streaming Trigger Graph)**.** *Considering a Temporal Datalog$^{\rightarrow}_{\ominus}$ program P, a stream S and a SEG G*

for P, we say that G is a Streaming Trigger Graph (STG) for $P \cup S$ if, for each fact a, we have $P \cup S \models a$ iff $a \in G(S)$. G is a STG for P if, for each stream S, G is a STG for $P \cup S$.

Towards practical STG-based materialisation, we introduce a form of temporal stratification that allows us to materialise Temporal Datalog$^{\rightarrow}_{\ominus}$ programs incrementally in time.

Definition 14 (Incremental Stratification)**.** *Consider a Temporal Datalog$^{\rightarrow}_{\ominus}$ program P, and the program P_T derived by removing from P all body conditions with time offset $k > 0$. A temporal stratification $\boldsymbol{B_1}, \boldsymbol{B_2}, \ldots$ of P is called an incremental stratification of P if, $\forall t \geq 1$, all tuples in the strata of $\boldsymbol{B_t}$ have time-point t, and the predicate sets obtained from $\boldsymbol{B_t}$ form a global stratification of P_T.*

The following corollary holds due to our restriction on forward-propagating and temporally stratified programs.

Corollary 1. *Every Temporal Datalog$^{\rightarrow}_{\ominus}$ program has an incremental stratification.*

To construct an incremental stratification $\boldsymbol{B_1}, \boldsymbol{B_2}, \ldots$ for program P, we compute a global stratification $\Sigma_1, \ldots, \Sigma_n$ for P_T, and set $\boldsymbol{B_t} = B_t^1, \ldots, B_t^n$ and $B_t^i = \{(p, t) | p \in \Sigma_i\}$, where $i \in [n]$, for each $t \geq 1$. We may then compute the perfect model of P over a stream S by setting $I^0 = S$ and processing $\boldsymbol{B_1}, \boldsymbol{B_2}, \ldots$ bottom-up. For each $\boldsymbol{B_t}$, we substitute T with t in P and evaluate its rules following the predicate stratification in $\boldsymbol{B_t}$, leading to interpretation I^t. Crucially, I^t is the perfect model of $P \cup S$ up to time-point t, enabling the incremental materialisation of P.

For TGs, k-compatibility ensures that, for each node v, its incoming edges stem from nodes defining the predicates in the conditions of $\mathrm{rule}(v)$ (see Definition 8). For STGs, we update k-compatibility to ensure that the time labels of these nodes are also consistent with the conditions of $\mathrm{rule}(v)$.

Definition 15 (k-compatibility)**.** *Consider program P with intensional rule r and SEG G. Tuple $(u_1, \ldots, u_n)$ of nodes from G is k-compatible with r at time t if:*

- *the predicate in the head of u_j equals the predicate of the j-th positive condition of r;*
- *$\mathrm{time}(u_j)$ is equal to t minus the offset in the j-th positive condition of r;*
- *each u_j was constructed at an iteration prior to k; and*
- *at least one u_j was constructed at iteration k - 1.*

Unfortunately, k-compatibility is inadequate when all tuple nodes were constructed when processing a previous stratum. Consider, e.g., this traffic light monitoring program:

$$red(L, T) \leftarrow green(L, T\text{ - }1). \quad (22)$$

$$red(L, T) \leftarrow green(L, T\text{ - }2). \quad (23)$$

$$green(L, T) \leftarrow red(L, T\text{ - }1), red(L, T\text{ - }2). \quad (24)$$

Given $red(a, 1)$ and $red(a, 2)$, we build an EG G^4 up to $t = 4$. In the first iteration, we build node for rule (24) with fact $green(a, 3)$. In the second iteration, we add node for rule (22) with fact $red(a, 4)$, completing materialisation up to $t = 4$. To update G^4 beyond $t = 4$, we need further iterations. However, since $green(a, 3)$ was computed two iterations ago, there are no k-compatible nodes with rule (23) at $t = 5$, prohibiting the derivation of $red(a, 5)$. To address this issue, we propose the notion of *consistency*.

Algorithm 4 STGmat

Input: A Temporal Datalog$^{\rightarrow}_{\ominus}$ program P and a stream S.
Output: The facts in an STG for $P \cup S$ up to time-point t.

1: $t := 0$, $k := 0$, $S_{\leq 0} := \emptyset$, $G := \emptyset$, $I^0 := \emptyset$
2: $\boldsymbol{B_t} :=$ IncrementallyStratify(P)
3: **while** True **do**
4: $t{:=}t+1$, $S_{\leq t}{:=}S_{\leq t\text{ - }1} \cup read(S, t)$, $P^t{:=}ground(P, t)$
5: **for all** $B_t^j \in \boldsymbol{B_t}$ **do**
6: **repeat**
7: $k := k + 1$, $I^k := I^{k\text{ - }1}$
8: **for all** rules r in P^t: $pred(head(r)) \in B_t^j$ **do**
9: **if** r is extensional **then**
10: G.add$((v, \emptyset, \emptyset, \mathrm{rule}(v) = r, \mathrm{time}(v) = t, \emptyset))$
11: **continue**
12: $E^- := \emptyset$
13: **for all** $b_i^- \in r$ **do**
14: $U := \{u | pred(head(\mathrm{rule}(u))) = pred(b_i^-) \wedge \mathrm{time}(u) = t\text{ - }offset(b_i^-)\}$
15: E^-.add$((U \rightarrow_i \{v\}))$
16: **if** $\forall b_i^+ \in r$: $pred(b_i^+) \notin B_t^j \vee offset(b_i^+) > 0$ **then**
17: **for all** $(u_1, \ldots, u_n)$ consistent with r **do**
18: G.add$((v, \bigcup_{i\in[n]}(u_i, v), E^-, \mathrm{rule}(v) = r, \mathrm{time}(v) = t, \bigcup_{i\in[n]} l((u_i, v)) = i))$
19: **else for all** $(u_1, \ldots, u_n)$ k-compatible with r **do**
20: G.add$((v, \bigcup_{i\in[n]}(u_i, v), E^-, \mathrm{rule}(v) = r, \mathrm{time}(v) = t, \bigcup_{i\in[n]} l((u_i, v)) = i))$
21: $G :=$ MinDatalog(G)
22: **for all** nodes v of iteration k **do** I^k.add$(v(S_{\leq t}, I^{k\text{ - }1}))$
23: **until** $I^k = I^{k\text{ - }1}$
24: $output(I^k)$, Forget$(G, S_{\leq t})$

Definition 16 (Consistency)**.** *Consider program P with intensional rule r and SEG G. Tuple $(u_1, \ldots, u_n)$ of nodes from G is consistent with r at time t if:*

- *the predicate in the head of u_j equals the predicate of the j-th positive condition of r;*
- *$\mathrm{time}(u_j)$ is equal to t minus the offset in the j-th positive condition of r; and*
- *each u_j was constructed at an iteration prior to k.*

Algorithm 4 outlines the steps of STG-based materialisation for program P and stream S. First, we compute an incremental stratification for P, with $\boldsymbol{B_t}$ denoting the strata with time-point t (line 2). Next, for each time-point t, we (i) read the facts of S occurring at t (line 4); (ii) ground the time variable of P to t, leading to program P^t (line 4); (iii) iterate over the strata B_t^j in $\boldsymbol{B_t}$ and update the STG G constructed thus far with nodes computing the predicates in B_t^j over P^t (lines 5–23); (iv) output the facts computed at t based on G (line 24); and (v) remove from G and $S_{\leq t}$—the stream read thus far—all nodes and extensional facts that cannot be used for derivations after t according to P (line 24).

For step (iii), we materialise the predicates in B_t^j in a process similar to TG-materialisation (compare lines 6–23 and Algorithm 1). The first key difference concerns negation: for each negative condition b_i^- in the rule of a node, we add one incoming n-edge whose source is the set of nodes with the same predicate as b_i^- and a time label correspond-

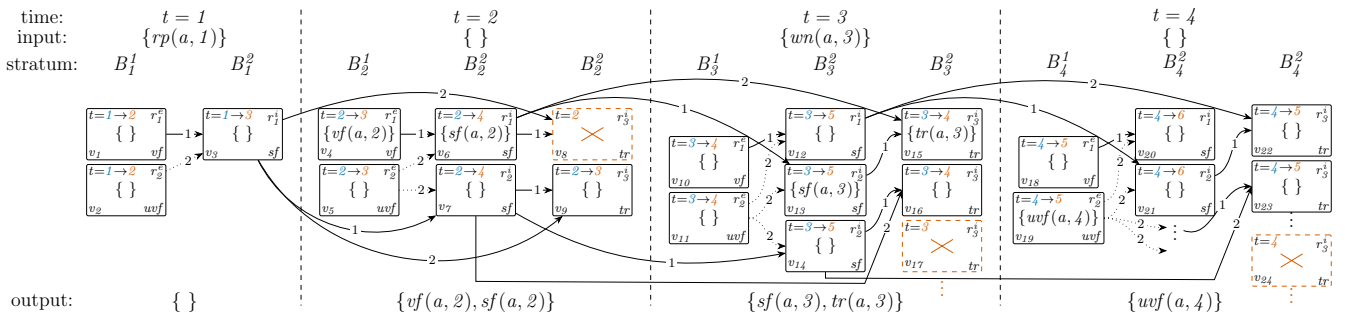

Figure 1: STG for Example 7 up to $t = 4$. For each node v, blue (resp. vermillion) coloured labels mark $\mathrm{time}(v)$ (the "forget time" of v).

ing to the time-point in b_i^- (lines 12–15). The second difference is that we rely on node consistency—instead of k-compatibility—for rules where all body conditions refer to a previous stratum, i.e., their predicate is not in B_t^j or their time-point is earlier than the current one (lines 16–18).

Optimisations. TGs reduce redundant computations via two optimisations (Tsamoura et al. 2021): (i) after each round k, nodes constructed at k whose query is subsumed by another node, based on rewritings recursively replacing intensional predicates with extensional ones following TG dependencies, are removed; and (ii) joins during node materialisation are restricted to tuples that are not in the previous interpretation. Both optimisations extend to STGs. For (i), n-edges stem from nodes in previous strata, so their facts can be treated as extensional in query rewritings (line 21). Optimisation (ii) applies unchanged (line 22). We may further optimise STGs by *forgetting* nodes whose facts can no longer contribute to future derivations based on their time-stamps.

Proposition 1 (Forgetting)**.** *Consider a Temporal Datalog$_\ominus^\rightarrow$ program P and an STG G for P with node v where* $\mathrm{rule}(v)$ *defines predicate p. If k_p is the maximum temporal offset of any body condition mentioning p across all rules in P, then v can be removed from G at evaluation time $t + k_p + 1$.*

Since k_p can be derived offline from P, forgetting adds no overhead to online materialisation.

Example 7 (STGmat)**.** *Consider the following program P, where rp, wn, vf, uvf, sf and tr abbreviate repair, warning, verified, unverified, safe and trusted.*

$$vf(X, T) \leftarrow rp(X, T\text{-}1). \qquad (r_1^e)$$
$$uvf(X, T) \leftarrow wn(X, T\text{-}1). \qquad (r_2^e)$$
$$sf(X, T) \leftarrow vf(X, T), \mathsf{not}\ uvf(X, T). \qquad (r_1^i)$$
$$sf(X, T) \leftarrow sf(X, T\text{-}1), \mathsf{not}\ uvf(X, T). \qquad (r_2^i)$$
$$tr(X, T) \leftarrow sf(X, T), sf(X, T\text{-}1). \qquad (r_3^i)$$

r_1^e and r_2^e are extensional, while r_1^i, r_2^i and r_3^i are intensional. Strata $B_t^1 = \{(rp, t), (wn, t), (vf, t), (uvf, t)\}$ and $B_t^2 = \{(sf, t), (tr, t)\}$ incrementally stratify P.

Figure 1 presents the STG constructed by Algorithm 4 over a stream with $rp(a, 1)$ and $wn(a, 3)$ up to $t = 4$. Starting from $t = 1$ and B_1^1, we first construct nodes v_1 and v_2 for extensional rules r_1^e and r_2^e, defining vf and uvf. These rules cannot fire at $t = 1$, since their conditions refer to the previous time-point, and thus v_1 and v_2 are empty. Moving to the next stratum B_1^2, v_1 is consistent with intensional rule r_1^i—defining sf—at time $t = 1$, leading to the creation of node v_3, with incoming p-edge $v_1 \rightarrow_1 v_3$ and n-edge $\{v_2\} \rightarrow_2 v_3$. No further nodes can be constructed for rules defining predicates in B_1^2, completing materialisation for $t = 1$. All conditions in P for predicates vf and uvf have temporal offset 0, while sf appears in conditions with offset 1. Thus, v_1 and v_2 are forgotten at the next time-step $t = 2$, while v_3 is forgotten when $t = 3$.

For $t = 2$ and B_2^1, r_1^e fires because of fact $rp(a, 1)$, populating node v_4 with $vf(a, 2)$. Since $wn(a, 1)$ is not in the input, $uvf(a, 2)$ cannot be proven, and thus an empty node v_5 is constructed for r_2^e. Moving to B_2^2, there are two distinct proofs for $sf(a, 2)$: the former requires $vf(a, 2)$ via r_1^i, while the latter requires $sf(a, 1)$ via r_2^i. Node v_6 captures the former proof via p-edge $v_4 \rightarrow_1 v_6$, and is populated with $sf(a, 2)$, while v_7 captures the latter proof. Next, we construct nodes v_8 and v_9, denoting the possible ways of proving $tr(a, 2)$. These proofs, however, are not distinct—given a proof of $sf(a, 1)$ via r_1^i, reducing $sf(a, 2)$ via r_1^i yields a subquery of the query produced through its reduction with r_2^i—and thus v_8 is removed before materialisation.

At $t = 3$, $sf(a, 3)$ is derived via r_2^i, and $tr(a, 3)$ then follows from r_3^i. At $t = 4$, $wn(a, 3)$ leads to $uvf(a, 4)$ in node v_{19}. The n-edge from v_{19} then blocks the derivation of $sf(a, 4)$ via both r_1^i and r_2^i, capturing their negative condition. The full trace up to $t = 4$ is shown in Figure 1.

Theorem 3 (STGmat Correctness)**.** *Algorithm 4 builds an STG G for Temporal Datalog$_\ominus^\rightarrow$ program P and stream S.*

Proof Sketch. We show by induction on the strata in $\boldsymbol{B_t}$ that $P \cup S \models a(\boldsymbol{x}, t)$ iff $a(\boldsymbol{x}, t) \in G(S)$. Correctness for B_1^1 follows from the correctness of TGmat (Algorithm 1), because the predicates in B_1^1 have no negative dependencies and their temporal dependencies are vacuous at $t = 1$. Assuming correctness up to B_1^{i-1}, we prove it for B_1^i: every negated condition "not b" in a rule r defining a predicate a where $(a, 1) \in B_1^i$ refers to a lower stratum, so $P \cup S \models b$ iff $b \in G(S)$ by the inductive assumption, while lines 13–15 add n-edges from all nodes that may contain b, ensuring correct evaluation. Assuming correctness up to B_{t-1}^n, correctness for B_t^1 follows from the consistency criterion (lines 16–18), which correctly materialises rules whose conditions all

refer to previous strata. The same arguments support the step from B_t^{i-1} to B_t^i, completing the proof. □

6 Summary, Related and Further Work

We proposed mappings from two prominent event specification languages (ESLs) for composite event recognition (CER), which are fragments of LARS and EC, to Temporal Datalog$_{\ominus}^{\rightarrow}$, i.e., a temporal Datalog with stratified negation and no future dependencies, and proved their correctness. We also introduced Streaming Trigger Graphs as an optimised evaluation mechanism for Temporal Datalog$_{\ominus}^{\rightarrow}$.

Numerous ESLs have been proposed for CER (Cugola and Margara 2010; Baumgartner 2021; Alevizos et al. 2024). DatalogMTL, e.g., combines Datalog with quantitative temporal modalities in a streaming setting (Wang et al. 2025). Contrary to LARS and EC, DatalogMTL is defined over a rational time model and uses intervals to quantify temporal offsets. DatalogMTL variants with forward-propagating rules (Walega et al. 2019), negation (Cucala et al. 2021) and an integer time model (Walega et al. 2020) have also been studied. We conjecture that such a restricted variant can be mapped to Temporal Datalog$_{\ominus}^{\rightarrow}$ in a similar way to our plain LARS translation, due to similarities between their temporal operators. Another prominent CER system is CORE (Grez et al. 2020; Bucchi et al. 2022). Contrary to Temporal Datalog$_{\ominus}^{\rightarrow}$, CORE is restricted to unary predicates, and thus cannot express relational composite events.

A LARS-to-ASP translation has been proposed for an incremental evaluation of LARS queries (Beck et al. 2017), but introduces multiple time variables per rule, producing programs outside Temporal Datalog$_{\ominus}^{\rightarrow}$. Furthermore, the stratification in (Beck et al. 2015) prohibits recursion via window operators in LARS, facilitating efficient reasoning. Our approach does not make this restriction.

Our work is motivated by earlier attempts to avoid "blocking queries", requiring arbitrary wait times on new streaming data (Babcock et al. 2002). Temporal Datalog$_{\ominus}^{\rightarrow}$ prohibits such queries by construction. Streamlog is also a temporal Datalog without blocking queries. Unlike our work, it allows equality constraints on time terms and reasons only over time-points appearing in the input (Das et al. 2018).

Several large-scale Datalog engines exist (Shkapsky et al. 2016; Ryzhyk and Budiu 2019; Ivliev et al. 2024), and incremental evaluation has been studied for streaming data (Budiu et al. 2024), but these approaches do not model temporal modalities. Temporal Vadalog supports temporal knowledge graph reasoning over existential rules (Bellomarini et al. 2025). Such rules have also been studied in the context of LARS (Urbani et al. 2022), DatalogMTL (Lanzinger et al. 2023), and trigger graphs (Tsamoura et al. 2021), thus constituting a natural future extension for STGs. Our work assumes ordered input, as is common in stream reasoning (Zaniolo 2012). Handling out-of-order streams (Tsilionis et al. 2022; Cruz-Filipe et al. 2025) via temporal delays in reasoning (Ronca et al. 2022) is left for future work. Finally, we aim to leverage a probabilistic extension of trigger graphs (Tsamoura et al. 2023) to support CER under uncertainty (Alevizos et al. 2017; Tiger and Heintz 2020; Tsilionis et al. 2025).

AI Declaration

No generative AI tools were used in the research, development, or writing of this paper.

Acknowledgements

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Alevizos, E.; Skarlatidis, A.; Artikis, A.; and Paliouras, G. 2017. Probabilistic complex event recognition: A survey. *Commun. ACM* 50(5):71:1–71:31.

Alevizos, E.; Artikis, A.; and Paliouras, G. 2024. Complex event recognition with symbolic register transducers. *Proc. VLDB Endow.* 17(11):3165–3177.

Artikis, A.; Sergot, M. J.; and Paliouras, G. 2015. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.* 27(4):895–908.

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In *PODS*, 1–16.

Baumgartner, P. 2021. Combining event calculus and description logic reasoning via logic programming. In *FroCoS*, 98–117.

Bazoobandi, H. R.; Beck, H.; and Urbani, J. 2017. Expressive stream reasoning with laser. In *ISWC*, volume 10587, 87–103.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2015. Answer update for rule-based stream reasoning. In *IJCAI*, 2741–2747.

Beck, H.; Eiter, T.; and Folie, C. 2017. Ticker: A system for incremental asp-based stream reasoning. *Theory Pract. Log. Program.* 17(5-6):744–763.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2018. LARS: A logic-based framework for analytic reasoning over streams. *Artif. Intell.* 261:16–70.

Bellomarini, L.; Sallinger, E.; and Gottlob, G. 2018. The vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.* 11(9):975–987.

Bellomarini, L.; Blasi, L.; Nissl, M.; and Sallinger, E. 2025. The temporal vadalog system: Temporal datalog-based reasoning. *Theory Pract. Log. Program.* 25(2):168–196.

Bucchi, M.; Grez, A.; Quintana, A.; Riveros, C.; and Vansummeren, S. 2022. CORE: a complex event recognition engine. *Proc. VLDB Endow.* 15(9):1951–1964.

Budiu, M.; Chajed, T.; McSherry, F.; Ryzhyk, L.; and Tannen, V. 2024. DBSP: incremental computation on streams and its applications to databases. *SIGMOD Rec.* 53(1):87–95.

Chomicki, J., and Imielinski, T. 1988. Temporal deductive databases and infinite objects. In *PODS*, 61–73. ACM.

Clark, K. L. 1977. Negation as failure. In *Logic and Data Bases*, 293–322. Plemum Press.

Cruz-Filipe, L.; Gaspar, G.; and Nunes, I. 2025. Can't you answer while you wait? *Ann. Math. Artif. Intell.* 93(5):727–758.

Cucala, D. J. T.; Walega, P. A.; Grau, B. C.; and Kostylev, E. V. 2021. Stratified negation in datalog with metric temporal operators. In *AAAI*, 6488–6495.

Cugola, G., and Margara, A. 2010. TESLA: A formally defined event specification language. In *DEBS*, 50–61. ACM.

Cugola, G., and Margara, A. 2012. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.* 44(3).

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3):374–425.

Das, A.; Gandhi, S. M.; and Zaniolo, C. 2018. ASTRO: A datalog system for advanced stream reasoning. In *CIKM*, 1863–1866.

Eiter, T., and Kiesel, R. 2020. Weighted LARS for quantitative stream reasoning. In *ECAI*, volume 325, 729–736.

Eiter, T.; Ogris, P.; and Schekotihin, K. 2019. A distributed approach to LARS stream reasoning (system paper). *Theory Pract. Log. Program.* 19(5-6):974–989.

Falcionelli, N.; Sernani, P.; de la Torre, A. B.; Mekuria, D. N.; Calvaresi, D.; Schumacher, M.; Dragoni, A. F.; and Bromuri, S. 2019. Indexing the event calculus: Towards practical human-readable personal health systems. *Artif. Intell. Medicine* 96:154–166.

Giatrakos, N.; Alevizos, E.; Artikis, A.; Deligiannakis, A.; and Garofalakis, M. N. 2020. Complex event recognition in the big data era: a survey. *VLDB J.* 29(1):313–352.

Grez, A.; Riveros, C.; Ugarte, M.; and Vansummeren, S. 2020. On the expressiveness of languages for complex event recognition. In *ICDT*, volume 155, 15:1–15:17.

Ivliev, A.; Gerlach, L.; Meusel, S.; Steinberg, J.; and Krötzsch, M. 2024. Nemo: Your friendly and versatile rule reasoning toolkit. In *KR*.

Khazael, B.; Vahidi-Asl, M.; and Malazi, H. T. 2023. Geospatial complex event processing in smart city applications. *Simul. Model. Pract. Theory* 122:102675.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Gen. Computing* 4(1):67–96.

Lanzinger, M.; Nissl, M.; Sallinger, E.; and Walega, P. A. 2023. Temporal datalog with existential quantification. In *IJCAI*, 3277–3285.

Mantenoglou, P., and Artikis, A. 2025. Temporal specification optimisation for the event calculus. In *AAAI*, 15075–15082.

Mantenoglou, P.; Pitsikalis, M.; and Artikis, A. 2022. Stream reasoning with cycles. In *KR*.

Mantenoglou, P.; Kelesis, D.; and Artikis, A. 2023. Complex event recognition with allen relations. In *KR*, 502–511.

Mantenoglou, P.; Pitsikalis, M.; and Artikis, A. 2025. Reasoning over streams of events with delayed effects. *JAIR* 84.

Montali, M.; Maggi, F. M.; Chesani, F.; Mello, P.; and van der Aalst, W. M. P. 2013. Monitoring business constraints with the event calculus. *ACM Trans. Intell. Syst. Technol.* 5(1):17:1–17:30.

Nenov, Y.; Piro, R.; Motik, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2015. Rdfox: A highly-scalable RDF store. In *ISWC*, volume 9367, 3–20.

Nomikos, C.; Rondogiannis, P.; and Gergatsoulis, M. 2005. Temporal stratification tests for linear and branching-time deductive databases. *Theor. Comput. Sci.* 342(2-3):382–415.

Palopoli, L. 1992. Testing logic programs for local stratification. *Theor. Comput. Sci.* 103(2):205–234.

Przymusinski, T. C. 1988. On the declarative semantics of deductive databases and logic programs. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 193–216.

Reiter, R. 1977. On closed world data bases. In *Logic and Data Bases*, 55–76. Plemum Press.

Ronca, A.; Kaminski, M.; Grau, B. C.; and Horrocks, I. 2018a. The window validity problem in rule-based stream reasoning. In *KR*, 571–581.

Ronca, A.; Kaminski, M.; Grau, B. C.; Motik, B.; and Horrocks, I. 2018b. Stream reasoning in temporal datalog. In *AAAI*, 1941–1948.

Ronca, A.; Kaminski, M.; Grau, B. C.; and Horrocks, I. 2022. The delay and window size problems in rule-based stream reasoning. *Artif. Intell.* 306:103668.

Ryzhyk, L., and Budiu, M. 2019. Differential datalog. *Datalog 2.0 Workshop in LPNMR* 2368:56–67.

Shkapsky, A.; Yang, M.; Interlandi, M.; Chiu, H.; Condie, T.; and Zaniolo, C. 2016. Big data analytics with datalog queries on spark. In *SIGMOD Conference*, 1135–1149.

Tiger, M., and Heintz, F. 2020. Incremental reasoning in probabilistic signal temporal logic. *Int. J. Approx. Reason.* 119:325 – 352.

Tsamoura, E.; Carral, D.; Malizia, E.; and Urbani, J. 2021. Materializing knowledge bases via trigger graphs. *Proc. VLDB Endow.* 14(6):943–956.

Tsamoura, E.; Lee, J.; and Urbani, J. 2023. Probabilistic reasoning at scale: Trigger graphs to the rescue. *Proc. ACM Manag. Data* 1(1):39:1–39:27.

Tsilionis, E.; Koutroumanis, N.; Nikitopoulos, P.; Doulkeridis, C.; and Artikis, A. 2019. Online event recognition from moving vehicles: Application paper. *Theory Pract. Log. Program.* 19(5-6):841–856.

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2022. Incremental event calculus for run-time reasoning. *J. Artif. Intell. Res.* 73:967–1023.

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2024. A tensor-based formalization of the event calculus. In *IJCAI*.

Tsilionis, E.; Artikis, A.; and Paliouras, G. 2025. A tensor-based probabilistic event calculus. In *KR*, 643–652.

Ullman, J. D. 1989. *Principles of Database and Knowledge-Base Systems, Volume II*. Computer Science Press.

Urbani, J.; Jacobs, C. J. H.; and Krötzsch, M. 2016. Column-oriented datalog materialization for large knowledge graphs. In *AAAI*, 258–264.

Urbani, J.; Krötzsch, M.; and Eiter, T. 2022. Chasing streams with existential rules. In *KR*, 415–419.

Walega, P. A.; Kaminski, M.; and Grau, B. C. 2019. Reasoning over streaming data in metric temporal datalog. In *AAAI*, 3092–3099.

Walega, P. A.; Grau, B. C.; Kaminski, M.; and Kostylev, E. V. 2020. Datalogmtl over the integer timeline. In *KR*, 768–777.

Walega, P. A.; Kaminski, M.; Wang, D.; and Grau, B. C. 2023. Stream reasoning with datalogmtl. *J. Web Semant.* 76:100776.

Wang, D.; Grau, B. C.; Walega, P. A.; and Hu, P. 2025. Practical reasoning in datalogmtl. *Theory Pract. Log. Program.* 25(2):225–255.

Zaniolo, C. 2012. Logical foundations of continuous query languages for data streams. In *Datalog in Academia and Industry*.

Zhang, H.; Diao, Y.; and Immerman, N. 2014. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD Conference*, 217–228. ACM.

Zielinski, B. 2023. Explanatory denotational semantics for complex event patterns. *Formal Aspects Comput.* 35(4):23:1–23:37.