On the Expressiveness of Languages for Complex Event Recognition

Alejandro Grez, Cristian Riveros, Martín Ugarte
PUC & IMFD Chile

Stijn Vansummeren
Université Libre de Bruxelles

Abstract

Complex Event Recognition (CER for short) has recently gained attention as a mechanism for detecting patterns in streams of continuously arriving event data. Numerous CER systems and languages have been proposed in the literature, commonly based on combining operations from regular expressions (sequencing, iteration, and disjunction) and relational algebra (e.g., joins and filters). While variables in these languages can only bind single elements, they also provide capabilities for filtering sets of events that occur inside iterative patterns; for example requiring sequences of numbers to be increasing. Unfortunately, these type of filters usually present ad-hoc syntax and under-defined semantics, precisely because variables cannot bind sets of events. As a result, CER languages that provide filtering of sequences commonly lack rigorous semantics and their expressive power is not understood.

In this paper we embark on two tasks: First, to define a denotational semantics for CER that naturally allows to bind and filter sets of events; and second, to compare the expressive power of this semantics with that of CER languages that only allow for binding single events. Concretely, we introduce Set-based Complex Event Logic (S-CEL for short), a variation of the CER language introduced in [18] in which all variables bind to sets of matched events. We then compare S-CEL with Event-based CEL (E-CEL), the language proposed in [18] where variables bind single events. We show that they are equivalent in expressive power when restricted to unary predicates but, surprisingly, incomparable in general. Nevertheless, we show that if we restrict to sets of binary predicates, then S-CEL is strictly more expressive than E-CEL. To get a better understanding of the expressive power, computational capabilities, and limitations of S-CEL, we also investigate the relationship between S-CEL and Complex Event Automata (CEA), a natural computational model for CER languages. We define a property on CEA called the *-property and show that, under unary predicates, S-CEL captures precisely the subclass of CEA that satisfy this property. Finally, we identify the operations that S-CEL is lacking to characterize CEA and introduce a natural extension of the language that captures the complete class of CEA under unary predicates.

2012 ACM Subject Classification Information systems → Data streams; Information systems → Query operators; Theory of computation → Formal languages and automata theory

Keywords and phrases Query languages, Complex Event Recognition, Logics, Automata theory

Digital Object Identifier 10.4230/LIPIcs.ICDT.2020.8

Acknowledgements A. Grez, C. Riveros and M. Ugarte were partially funded by the Millennium Institute for Foundational Research on Data.

1 Introduction

The timely processing of data streams, where new data is continuously arriving, is a key ingredient of many contemporary Big Data applications. Examples of such applications include the recognition of: attacks in computer networks [9, 10]; human activities in video content [19]; traffic incidents in smart cities [4]; and opportunities in the stock market [21]. Numerous systems for processing streaming data have been proposed over the decades (see, e.g., [12, 20] for surveys). Complex Event Recognition (CER for short) systems are specialized

type	T	H	H	T	H	T	H	H	T	H	...
value	-2	30	20	-1	27	2	45	50	-2	65	...
index	0	1	2	3	4	5	6	7	8	9	...

Figure 1 A stream S of events measuring temperature (T) in Celsius degrees and humidity (H) as a percentage of water in the air.

stream processing systems that allow to detect higher-level *complex events* from streams of simple *events*. In CER systems, users write so-called *patterns* that describe the sequences of simple events that trigger the recognition of a complex event.

To support the above-mentioned application scenarios, several CER systems and languages have been proposed in the literature—see e.g., the surveys [3, 12] and the references therein. Most notably, CER is supported by several contemporary Big Data streaming engines, such as Trill [8] and Flink [6]. However, as noted in [12], the literature focuses mostly on the practical aspects of CER, resulting in many heterogeneous implementations with fundamentally different capabilities. As a result, little is known on the formal foundations of CER and, in contrast to the situation for relational databases, we currently lack a common understanding of the trade-offs between expressiveness and complexity in the design of CER languages, as well as an established theory for optimizing CER patterns.

Towards a better understanding of the formal foundations of CER, a subset of the authors has recently proposed and studied a formal logic that captures the core features found in most CER languages [18]. This logic, which we will call Event-based Complex Event Logic or simply E-CEL, combines the regular expression operators (sequencing, to require that some pattern occurs before another; iteration, to recognize a pattern a number of times; and disjunction) with data filtering features as well as limited data outputting capabilities. E-CEL follows the approach that seems to be taken by most of the CER literature (e.g., [1, 11, 14, 15, 25], see also [12, 20]) in that data filtering is supported by binding variables to individual events in the stream, which can later be inspected by means of one or more predicates.

One of the main contributions of E-CEL is to provide formal semantics for a language that combines filtering capabilities with iteration. In particular, a challenging yet common task in CER systems is to filter variables occurring inside a Kleene closure in a wider scope, stating properties that involve all the events *captured* in different iterations. For this reason, variables that bind single events interact rather awkwardly with Kleene closure. Indeed, if a variable occurs inside a Kleene closure operator, what does the variable refer to when used outside this operator? In many of the practical CER languages, variables that bind single events are used inside Kleene Closure to express properties on sequences of events rather than on individual events, making the semantics confusing and unnatural.

To illustrate this semantics issue, let us introduce the following running example. Suppose that sensors are positioned in a farm to detect freezing plantations. Sensors detect temperature and humidity, generating a stream of events of two types, T and H, both of which have a *value* attribute that contains the measured temperature or humidity, respectively. We encode each event as a relational tuple, and an event stream is an infinite sequence of events. Furthermore, events are assumed to appear in generation order in the stream. Figure 1 shows an example, where *index* marks the position of the event in the stream.

Suppose now that a farmer is interested in checking events of freezing plantations. One possible specification representing this could be the following:

> "*after having a temperature below 0 degrees, there is a period where humidity increases until humidity is over* 60%".

To motivate the semantics mismatch between event-based variables and more general filters in CER languages, let us consider how one can define this complex event with two of the most influential CER languages in the literature [12], namely Cayuga [13–15] and SASE [1, 24, 25]. The CER languages of contemporary big data systems such as Trill [8] and Flink [6] are based on the former, and are therefore prone to the same mismatch.

1. In Cayuga, this complex event can be defined as follows:

$$\begin{aligned}&\texttt{FILTER}\{value < 0\}\ T\\ &\qquad\qquad \texttt{FOLD}\big\{\$2.value < \$.value,\ \$2.value \geq \$.value \ \texttt{AND}\ \ \$2.value \geq 60\big\}\ H \qquad (1)\end{aligned}$$

   Here, the subexpression ($\texttt{FILTER}\ \{value < 0\}\ T$) takes the stream of all events of type T and produces a new stream only with those events satisfying $value < 0$. Then, this output stream is processed by the `FOLD` operator. A stream expression of the form $S1\ \texttt{FOLD}\{\mathit{filter\_next}, \mathit{filter\_stop}\}\ S2$ is processed as follows[1]. Every time you receive an event from the stream $S1$, start collecting all elements from the stream $S2$ that satisfy *filter_next*, until you see an element from the stream $S2$ satisfying *filter_stop*. This allows to perform some incremental computations and variables '\$2' and '\$' refer to the previous and current iteration, respectively. In our Cayuga query, $\$2.value\ <\ \$.value$ is checking that we see an increasing sequence of H values, until we see the last non-increasing H value that is over 60% (i.e. $\$2.value \geq \$.value\ \texttt{AND}\ \$2.value \geq 60$).
2. In SASE, we can define the complex event more directly with the following query:

$$\begin{aligned}&\texttt{PATTERN}\,\text{SEQ}(T\ t,\ \ H^{+}\ h1[],\ \ H\ h2)\\ &\qquad\quad \texttt{WHERE}\ t.value < 0\ \ \texttt{AND}\ \ h1[i-1].value < h1[i].value\ \texttt{AND}\ \ h2.value \geq 60 \qquad (2)\end{aligned}$$

   The query looks for a T event (t), followed by one or more H events (collectively called $h1[]$), followed by a final H event ($h2$). The query then states that t's value is below zero, that subsequent events in $h1[]$ have increasing values, and $h2$'s value is above 60.

The two previous queries motivate the aforementioned ad-hoc semantics where events and sets of events are somehow intermingled. Cayuga uses event variables (e.g. \$ and \$2) to define, in a procedural way, a property over the set of H events. Instead, the SASE query implicitly combines event variables (e.g. t and $h2$) with set variables (e.g. $h1[]$). Indeed, SASE uses the ad-hoc notation $h1[i-1].value < h1[i].value$ to declare a predicate over a set of events captured by $h1[]$. It is important to mention that in both languages the semantics of the sets is not formally defined and, moreover, sets are actually not acknowledged as such. As a consequence, CER languages are designed without a good understanding of the implications of using event variables versus set variables, or how to compare them with other formalisms proposed in the literature. Moreover, while both Cayuga and SASE propose a computational model based on automata for evaluating queries, the relationship in expressive power between the CER language and their computational models has not been studied.

In this paper, we embark on the task of understanding the expressive power of CER languages that only allow binding and filtering individual events versus those that allow binding and filtering sets of events, as well as their corresponding computational models. Concretely we consider E-CEL [18] as a model of the former class of languages, and we

[1] Cayuga's `FOLD` operator actually also takes a third parameter that specifies the event to be output once the termination condition is met; for the sake of simplification we omit this parameter here.

introduce Set-based Complex Event Logic (S-CEL for short) as a model for the second class of languages. Variables in S-CEL can only bind and filter sets of matched events.

Specifically, we compare E-CEL against S-CEL and show that they are equivalent in expressive power when equipped with the same unary predicates but, surprisingly, incomparable when equipped with n-ary predicates, $n > 1$. In particular, when equipped with sets of binary predicates, S-CEL is strictly more expressive than E-CEL. However, when equipped with sets of ternary predicates, the languages are incomparable. The intuition behind this is that S-CEL cannot distinguish the events captured by a single variable inside a Kleene closure, while this is possible in E-CEL by using a clever trick that relies on ternary filters (Section 4).

Since E-CEL and S-CEL coincide when they are restricted to unary predicates, we study the expressiveness of this core CER language and compare it with a computational model for detecting complex events called *Complex Event Automata* [18] (CEA for short). We show that, in this setting, E-CEL and S-CEL are strictly weaker than CEA, but capture the subclass of CEA that satisfy the so-called $*$-property. Intuitively, this property indicates that the CEA can only make decisions based on events that are part of the output. As a by-product of our development we are able to show that certain additional CER operators that have been proposed in the literature, such as `AND` and `ALL`, do not add expressive power to E-CEL and S-CEL while others, such as `UNLESS`, provide the languages with new capabilities (Section 5).

Finally, we identify the operations that S-CEL lacks to capture CEA and introduce a natural extension that captures the complete class of CEA under unary predicates. This is the first time that a CER language is proposed to capture the full expressive power of its underlying computational model. As a result we are also able to give insight into the `STRICT` selection policy and strict operator that are usually supported by CER languages (Section 6).

Related Work. As already mentioned, the focus in the majority of the CER literature is on the systems aspects of CER rather than on the foundational aspects, and there is no formal study of the expressiveness of CER languages. A notable exception is the work by Zhang et al on SASE$^+$ [25], which considers the descriptive complexity of a core CER language. The syntax and semantics of SASE$^+$ are defined in a technical report [16], but unfortunately they are underdefined. Moreover, the semantics is not denotational but operational: the paper describes how to compile a subset of the language into a computational model, and defines the semantics simply as the output of the resulting instance.

Extensions of regular expressions with data filtering capabilities have been considered outside of the CER context. *Extended regular expressions* [2, 5, 7] extend the classical regular expressions operating on strings with variable binding expressions of the form $x\{e\}$ (meaning that when the input is matched, the substring matched by regular expression e is bound to variable x) and variable backreference expression of the form $\&x$ (referring to the last binding of variable x). Variables binding expressions can occur inside a Kleene closure, but when referred to, a variable always refers to the last binding. Extended regular expressions differ from S-CEL and E-CEL in that they operate on finite strings over a finite alphabet rather than infinite streams over an infinite alphabet of possible events; and use variables only to filter the input rather than also using them to construct the output. Regular expressions with variable bindings have also been considered in the so-called spanners approach to information extraction [17]. There, however, variables are only used to construct the output and cannot be used to inspect the input. In addition, variable binding inside Kleene closures is prohibited.

Languages with variables binding sets, such as monadic second order logic (MSO), are standard in logic and databases [22]. However, we are not aware of any CER language such as S-CEL that combines regular operators with variables that bind sets of events.

2 Preliminaries

In this section we introduce the formal definitions for streams and complex events, and recall the definition of E-CEL, as introduced in [18].

Schemas, Tuples and Streams. Let $\mathbf{A}$ be an infinite set of *attribute names* and $\mathbf{D}$ an infinite set of values. A database schema $\mathcal{R}$ is a finite set of relation names, where each relation name $R \in \mathcal{R}$ is associated to a tuple of attributes denoted by $\text{att}(R)$. If R is a relation name, then an R-tuple is a function $t : \text{att}(R) \to \mathbf{D}$. We say that the type of an R-tuple t is R, and denote this by $\text{type}(t) = R$. For any relation name R, $\text{tuples}(R)$ denotes the set of all possible R-tuples. Similarly, for any database schema $\mathcal{R}$, $\text{tuples}(\mathcal{R}) = \bigcup_{R\in\mathcal{R}} \text{tuples}(R)$. Given a schema $\mathcal{R}$, an $\mathcal{R}$*-stream* S is an infinite sequence $S = t_0t_1\ldots$ where $t_i \in \text{tuples}(\mathcal{R})$. When $\mathcal{R}$ is clear from the context, we refer to S simply as a stream. Given a stream $S = t_0t_1\ldots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$, and the sub-stream $t_it_{i+1}\ldots$ is denoted by S_i. We consider in this paper that the time of each event is given by its index, and defer a more elaborated time model (like [23]) to future work.

E-CEL syntax. Let $\mathbf{X}$ be a set of variables. Given a schema $\mathcal{R}$, an event predicate of arity n is an n-ary relation P over $\text{tuples}(\mathcal{R})$, $P \subseteq \text{tuples}(\mathcal{R})^n$. If $\mathcal{P}$ is a set of event predicates then an atom over $\mathcal{P}$ is an expression $P(x_1,\ldots,x_n)$ with $P \in \mathcal{P}$ of arity n and $x_1,\ldots,x_n$ variables in $\mathbf{X}$. The set of formulas of E-CEL($\mathcal{P}$) over schema $\mathcal{R}$ is given by the grammar:

$$\varphi \;:=\; R \;\texttt{AS}\; x \;\mid\; \varphi \;\texttt{FILTER}\; P(\bar{x}) \;\mid\; \varphi \;\texttt{OR}\; \varphi \;\mid\; \varphi\,;\varphi \;\mid\; \varphi+ .$$

Here, R ranges over relation names in $\mathcal{R}$, x over variables in $\mathbf{X}$ and $P(\bar{x})$ over $\mathcal{P}$.

E-CEL semantics. For the semantics of E-CEL we first need to introduce the notion of *complex event*. A complex event C is defined as a non-empty and finite set of natural numbers. We denote by $\min(C)$ and $\max(C)$ the minimum and maximum element of C, respectively. Given two complex events C_1 and C_2, we write $C_1 \cdot C_2$ for their *concatenation*, which is defined as $C_1 \cdot C_2 := C_1 \cup C_2$ whenever $\max(C_1) < \min(C_2)$ and empty otherwise. Given an E-CEL formula φ, we denote by $\text{vdef}(\varphi)$ all variables defined in φ by a clause of the form R AS x and by $\text{vdef}_+(\varphi)$ all variables in $\text{vdef}(\varphi)$ that are defined outside the scope of all $+$-operators. For example, in the formula:

$$\varphi \;=\; (T \;\texttt{AS}\; x\,;\,(H \;\texttt{AS}\; y \;\texttt{FILTER}\; y.id = x.id)+;(T \;\texttt{AS}\; z)+) \;\texttt{FILTER}\; (u.id = 1)$$

we have $\text{vdef}(\varphi) = \{x, y, z\}$ and $\text{vdef}_+(\varphi) = \{x\}$. Note that in this formula the variable u is only mentioned in the filter, and is therefore somehow *unsafe*. The problem of unsafe variables in CER is recurrent when trying to correlate variables occurring outside a Kleene closure with variables occurring inside a Kleene closure. We refer the interested reader to the extended safeness discussion in [18]. A valuation is a function $\nu : \mathbf{X} \to \mathbb{N}$. Given a finite subset $U \subseteq \mathbf{X}$ and two valuations ν_1 and ν_2, we define the valuation $\nu_1[\nu_2/U]$ by $\nu_1[\nu_2/U](x) = \nu_2(x)$ whenever $x \in U$ and $\nu_1[\nu_2/U](x) = \nu_1(x)$ otherwise.

Now we are ready to define the semantics of E-CEL. Given an E-CEL-formula φ, we say that a complex event C belongs to the evaluation of φ over a stream S starting at position i, ending at position j, and under the valuation ν (denoted by $C \in [\![\varphi]\!](S, i, j, \nu)$) if $i \le j$ and one of the following conditions holds:

- $\varphi = R$ AS x, $C = \{\nu(x)\}$, $\text{type}(S[\nu(x)]) = R$ and $i \le \nu(x) = j$.

- $\varphi = \rho$ `FILTER` $P(x_1, \ldots, x_n)$, and both $C \in [\![\rho]\!](S, i, j, \nu)$ and $(S[\nu(x_1)], \ldots, S[\nu(x_n)]) \in P$ hold.
- $\varphi = \rho_1$ `OR` ρ_2, and $C \in [\![\rho_1]\!](S, i, j, \nu)$ or $C \in [\![\rho_2]\!](S, i, j, \nu)$.
- $\varphi = \rho_1 \,;\, \rho_2$ and there exists $k \in \mathbb{N}$ and complex events C_1 and C_2 such that $C = C_1 \cdot C_2$, $C_1 \in [\![\rho_1]\!](S, i, k, \nu)$ and $C_2 \in [\![\rho_2]\!](S, k+1, j, \nu)$.
- $\varphi = \rho+$ and $C \in \bigcup_{k=1}^{\infty} [\![\rho[k]]\!](S, i, j, \nu)$ where $C \in [\![\rho[k]]\!](S, i, j, \nu)$ if there exists a valuation ν' such that either $C \in [\![\rho]\!](S, i, j, \nu[\nu'/U])$ if $k = 1$ or $C \in [\![\rho \,;\, \rho[k-1]]\!](S, i, j, \nu[\nu'/U])$ otherwise, where $U = \text{vdef}_+(\rho)$.

We say that C belongs to the evaluation of φ over S at position $n \in \mathbb{N}$, denoted by $C \in [\![\varphi]\!]_n(S)$, if $C \in [\![\varphi]\!](S, 0, n, \nu)$ for some valuation ν. Notice that the definition of E-CEL in [18] did not use the bounds i and j. We use them here just for consistency with the other definitions in the paper (S-CEL and S-CEL+). Note also that since $\nu(x) = j$ in the first bullet, the event in position j will always be part of the matched complex event. The reason behind this design decision is that one would like to be able to produce an output as soon as the final event participating in the output is processed.

▶ **Example 1.** Consider that we want to use E-CEL to see how temperature changes at some location whenever there is an increase of humidity from below 30 to above 60. Assume, for this example, that the location of an event (i.e. the location of a sensor) is recorded in its *id* attribute. Then, using a self-explanatory syntax for predicates, we would write:

$$[H \texttt{ AS } x \,;\, (T \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+ \,;\, H \texttt{ AS } z]$$
$$\texttt{FILTER } (x.value < 30 \;\wedge z.value > 60 \wedge x.id = z.id)$$

Inside the Kleene closure, y is always bound to the current event being inspected. The filter $y.id = x.id$ ensures that the inspected temperature events are of the same location as the first humidity event x. Note that, in this case, the output is a complex event, and includes in particular the positions of the inspected T events.

3 Set-based Complex Event Logic

In this section, we formally define S-CEL, a core complex event recognition language in which all variables bind complex events instead of individual events. Before giving the formal definition, we first give a gentle introduction to S-CEL and the design decisions behind its syntax and semantics.

As discussed in the Introduction, practical CER languages use variables that bind both single events (e.g. '\$' in (1) and t in (2)) and complex events (e.g. $h[]$ in (2)). In S-CEL variables bind to complex events, and predicates are over complex events instead of individual events. As an example, recall our statement for detecting freezing plantations: "*after having a temperature below 0 degrees, there is a period where humidity increases until humidity is over* 60%". This statement can be defined in S-CEL with the following formula:

$$\varphi \;=\; T; (H+ \texttt{ IN } HS); (H \texttt{ IN } LH) \texttt{ FILTER } (T.value < 0 \wedge \text{incr}(HS) \wedge LH.value \geq 60) \quad (3)$$

To understand the meaning of this formula, note that T and H are relation names while HS (Humidity Sequence) and LH (Last Humidity) are variables. These two variables, HS and LH, are assigned to the complex events defined by the subformulas $H+$ and H, respectively, by using the `IN`-operator. For example, if $\{4, 6, 7\}$ is a complex event defined by $H+$ (i.e. a sequence of one or more H-events) then HS will be assigned to $\{4, 6, 7\}$. We denote this as

$HS \to \{4, 6, 7\}$. Similarly, if the subformula H (i.e. only one H event) defines the complex event $\{9\}$, then $LH \to \{9\}$. Strictly speaking LH represents a complex event, although because of the pattern it will always contain only a single event. Note that T is not assigned to any variable in φ despite that we later used T in the filter clause. In S-CEL we use relational names themselves also as variables; this generally decreases the number of variables in a formula and aids readability. Thus, T is also used as a variable in φ and $T \to \{3\}$ is a valid assignment of the T-events.

Now that all variables are assigned to complex events, we can check that they respect the order imposed by the sequencing operator (;): $T \to \{3\}$ is followed by the sequence $HS \to \{4, 6, 7\}$, which is followed by $LH \to \{9\}$. All together they form the complex event $C = \{3, 4, 6, 7, 9\}$. Indeed, variables T, HS and LH are assigned to the relevant part C which are used in the filter clause to check through built-in predicates that they satisfy the required conditions: (1) the temperature is below 0, (2) the humidity forms an increasing sequence and (3) the last humidity is over 60%. The first and third properties are naturally checked with the predicates $T.value < 0$ and $LH.value \geq 60$. The second property can be checked through a S-CEL predicate that restricts the complex event in HS to form an increasing sequence of humidity values (similar to the predicate $h1[i-1].value < h1[i].value$ in (2)).

In S-CEL we allow to use arbitrary predicates over complex events. This might seem too relaxed at first, as predicates could specify arbitrary properties. However, the goal of this approach is to separate what is inherent to a CER framework and what is particular to an application. In particular, each application is free to choose any set of predicates that can be useful and meaningful for users, as well as the algorithms and evaluation strategies to evaluate them. Next, we give the syntax and semantics of S-CEL.

S-CEL syntax. Let $\mathbf{L}$ be a finite set of variables containing all relation names (i.e. $\mathcal{R} \subseteq \mathbf{L}$). A set predicate of arity n is an n-ary relation P over sets of tuples, $P \subseteq (2^{\text{tuples}(\mathcal{R})})^n$. We write arity$(P)$ for the arity of P. Let $\mathcal{P}$ be a set of set predicates. An atom over $\mathcal{P}$ is an expression of the form $P(A_1, \ldots, A_n)$ where $P \in \mathcal{P}$ is a predicate of arity n, and $A_1, \ldots, A_n \in \mathbf{L}$ (we also write $P(\bar{A})$ for $P(A_1, \ldots, A_n)$). The set of formulas in S-CEL($\mathcal{P}$) is given by the following syntax:

$$\varphi \;:=\; R \;\mid\; \varphi \texttt{ IN } A \;\mid\; \varphi \texttt{ FILTER } P(\bar{A}) \;\mid\; \varphi \texttt{ OR } \varphi \;\mid\; \varphi \,;\, \varphi \;\mid\; \varphi+$$

where R ranges over relation names, A over labels in $\mathbf{L}$ and $P(\bar{A})$ over $\mathcal{P}$.

S-CEL semantics. A set valuation (or just valuation if clear from the context) is a function $\mu : \mathbf{L} \to 2^{\mathbb{N}}$ such that $\mu(A)$ is a finite set for every $A \in \mathbf{L}$. The support of such a valuation is defined as $\text{supp}(\mu) = \bigcup_{a \in \mathbf{L}} \mu(A)$. Given two valuations μ_1 and μ_2, their union is defined by $(\mu_1 \cup \mu_2)(A) = \mu_1(A) \cup \mu_2(A)$ for every $A \in \mathbf{L}$. Finally, given a complex event C we define $S[C] = \{S[i] \mid i \in C\}$, namely, the set of tuples in S positioned at the indices specified by C.

Now we are ready to define the semantics of S-CEL formulas. Given a S-CEL formula φ, a stream S, and positions $i \leq j$, we say that a complex event C belongs to the evaluation of φ over a stream S starting at position i and ending at position j, and under the set valuation μ (denoted by $C \in [\![\varphi]\!](S, i, j, \mu)$) if one of the following conditions holds:

- $\varphi = R$, $C = \mu(R) = \{j\}$, type$(S[j]) = R$ and $\mu(A) = \emptyset$ for every $A \neq R$.
- $\varphi = \rho \texttt{ IN } A$, $\mu(A) = C$, and there exists a valuation μ' such that $C \in [\![\rho]\!](S, i, j, \mu')$ and $\mu(B) = \mu'(B)$ for all $B \neq A$. Intuitively, μ extends μ' assigning C to A.
- $\varphi = \rho \texttt{ FILTER } P(A_1, \ldots, A_n)$, and both $C \in [\![\rho]\!](S, i, j, \mu)$ and $(S[\mu(A_1)], \ldots, S[\mu(A_n)]) \in P$ hold.
- $\varphi = \rho_1 \texttt{ OR } \rho_2$ and $C \in [\![\rho_1]\!](S, i, j, \mu)$ or $C \in [\![\rho_2]\!](S, i, j, \mu)$.

- $\varphi = \rho_1 ; \rho_2$ and there exists $k \in \mathbb{N}$, complex events C_1 and C_2, and valuations μ_1 and μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in [\![\rho_1]\!](S, i, k, \mu_1)$ and $C_2 \in [\![\rho_2]\!](S, k+1, j, \mu_2)$.
- $\varphi = \rho+$, and $C \in \bigcup_{k=1}^{\infty} [\![\rho^k]\!](S, i, j, \mu)$ where $\rho^k = \rho; \cdots ; \rho$ k-times.

Observe that, by definition, if $C \in [\![\varphi]\!](S, i, j, \mu)$ then C is a subset of $\{i, \ldots, j\}$ and $j \in C$. Furthermore, one can easily show by induction over the size of φ that the support of μ is equal to C, namely, $C = \text{supp}(\mu)$. Similar to E-CEL we say that C belongs to the evaluation of a S-CEL formula φ over S at position $n \in \mathbb{N}$, denoted by $C \in [\![\varphi]\!]_n(S)$, if $C \in [\![\varphi]\!](S, 0, n, \mu)$ for some set valuation μ.

▶ **Example 2.** Consider the formula φ in (3) that detects possible freezing plantations. We illustrate the semantics of φ over the stream S depicted in Figure 1 where event types T and H have both a *value* attribute and an *index* attribute recording their index in the stream.

First, note that although conjunction of predicates is not directly supported in S-CEL, this can be easily simulated by a nesting of filter operators. Then, for the sake of simplification, we can analyze φ by considering each filter separately. For the subformula $\varphi_T = T$ FILTER $T.value < 0$ we can see that (i) $\{3\} \in [\![\varphi_T]\!](S, 0, 3, \mu_1)$ with $\mu_1(T) = \{3\}$. On the other hand, the last event (i.e. 9) is the only event that satisfies $\varphi_H = (H$ IN $LH)$ FILTER $LH.value \geq 60$ and then (ii) $\{9\} \in [\![\varphi_H]\!](S, 8, 9, \mu_2)$ with $\mu_2(LH) = \mu_2(H) = \{9\}$.

Now, the intermediate formula $\varphi_+ = (H+$ IN $HS)$ FILTER $\text{incr}(HS)$ captures a sequence of one or more H-events representing an increasing sequence of humidities. Because Kleene closure allows for arbitrary events to occur between iterations, these sequences can be selected from the powerset of all H-events that produced an increasing sequence like, for example, $\{4, 6, 7\}$ or $\{2, 4\}$. In particular, we have that (iii) $\{4, 6, 7\} \in [\![\varphi_+]\!](S, 4, 7, \mu_3)$ with $\mu_3(LH) = \mu_2(H) = \{4, 6, 7\}$. Putting together (i), (ii) and (iii) and noticing that $\varphi = \varphi_T; \varphi_+; \varphi_H$, we have that $\{3, 4, 6, 7, 9\} \in [\![\varphi]\!](S, 0, 9, \mu)$ with $\mu = \mu_1 \cup \mu_2 \cup \mu_3$. Finally, we remove μ and $\{3, 4, 6, 7, 9\}$ is a complex event in $[\![\varphi]\!]_9(S)$.

The reader might find the semantics of S-CEL more flexible and simpler than the one of E-CEL: the assignment of variables is more flexible and the semantics of iteration simpler (since variables are not re-assigned). We argue that the reason for this relies on the use of event-binding variables in order to manage sets (i.e. complex events). For example, in E-CEL variables can only be assigned at the event definition, with the atomic formula R AS x. In contrast, variables in S-CEL can manage complex events, allowing to use the IN-operator anywhere in a formula. Another more interesting example is iteration. In order to use event variables in a formula of the form $\varphi+$ we are forced to reassign these variables every time the subformula φ is evaluated (i.e. the use of the valuation $\nu_1[\nu_2/U]$). On the other hand, set valuations can naturally be merged by union (i.e. $\mu_1 \cup \mu_2$) and, therefore, the iteration is just a simple generalization of the sequencing operator (;).

It is important to notice that it is possible to define a more general language CEL that includes event and set variables. Given that in this paper our expressiveness analysis is always between E-CEL and S-CEL, we decide to present both language separately. We leave for future work the study of a CER query language that includes both approaches.

4 Set Variables Versus Event Variables

In this section, we compare the expressiveness of E-CEL and S-CEL. Since in traditional logics languages with variables binding sets (like MSO) can usually encode all formulas of a corresponding language binding single elements (like FO), this could suggest that S-CEL is more expressive than E-CEL. We show that this is only partially true: S-CEL includes E-CEL for binary predicates but they are incomparable in general.

In order to make a fair comparison between E-CEL and S-CEL we first need to agree on how we relate the event predicates that can be used in E-CEL to the set predicates that can be used in S-CEL. Indeed, the expressive power of both languages inherently depends on the allowed predicates, and we need to put them on equal ground in this respect. In particular, without any restrictions on the predicates of S-CEL we can easily express formulas that are beyond the scope of E-CEL. For this reason, we will restrict ourselves to set predicates created as *extensions* of event predicates. Given an event predicate $P(x_1, \ldots, x_n)$, we define its *set-extension* P^{S} to be the set predicate of the same arity as P such that $(S_1, \ldots, S_n) \in P^{\mathrm{S}}$ iff $\forall x_1 \in S_1, \ldots, x_n \in S_n$ it is the case that $(x_1, \ldots, x_n) \in P$. We extend this definition to sets of predicates: if $\mathcal{P}$ is a set of event predicates, $\mathcal{P}^{\mathrm{S}}$ is the set $\{P^{\mathrm{S}} \mid P \in \mathcal{P}\}$. In what follows we will compare E-CEL($\mathcal{P}$) to S-CEL($\mathcal{P}^{\mathrm{S}}$).

▶ **Example 3.** Using the set-extensions of the unary event predicates (e.g. $X.value < 30 := \forall x \in X\ x.value < 30$) and the binary id-comparison predicate (e.g. $X.id = Y.id := \forall x \in X \forall y \in Y\ x.id = y.id$), the E-CEL expression of Example 1 can be written in S-CEL as:

$$(H \texttt{ IN } X; (T+ \texttt{ IN } Y); H \texttt{ IN } Z) \texttt{ FILTER}$$
$$(X.value < 30 \wedge Z.value > 60 \wedge X.id = Y.id \wedge X.id = Z.id).$$

One could ask why do we focus on *universal* extensions of event predicates. After all, one could also consider *existential* extensions of the form $P^{\exists}$ where $(S_1, \ldots, S_n) \in P^{\exists}$ iff $\exists x_1 \in S_1, \ldots, x_n \in S_n.\ (x_1, \ldots, x_n) \in P$. Under this notion, S-CEL cannot meaningfully filter events captured by a Kleene closure. For example, if $X.\mathtt{id} = Y.\mathtt{id}$ is used with an existential semantics in Example 3, it would include in Y the T events occurring between the first H event and the second H event, as long as there is one such T event with the corresponding id. Therefore, although existential extensions could be useful in some particular CER use-cases, we compare E-CEL with S-CEL by considering only universal extensions.

We now compare both languages, considering the arity of the allowed predicates. We start by showing that if $\mathcal{U}$ is a set of unary event predicates, E-CEL($\mathcal{U}$) and S-CEL($\mathcal{U}^{\mathrm{S}}$) have the same expressive power. Formally, we say that two formulas ψ and φ are equivalent, denoted by $\psi \equiv \varphi$, if $[\![\psi]\!]_n(S) = [\![\varphi]\!]_n(S)$ for every stream S and position n.

▶ **Theorem 4.** *Let $\mathcal{U}$ be any set of unary event predicates. For every formula $\psi \in$ E-CEL($\mathcal{U}$) there exists a formula $\varphi \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$) such that $\psi \equiv \varphi$, and vice versa.*

The previous theorem is of particular relevance since it shows that both languages coincide in a well-behaved core. E-CEL with unary predicates was extensively studied in [18] showing efficient evaluation algorithms and it is part of almost every CER language [12].

Now we show that if we go beyond unary predicates there are S-CEL formulas that cannot be equivalently defined in E-CEL (under the same set of predicates). Let $\mathcal{P}_=$ be the smallest set of event predicates that allows to express equality between attributes of tuples and is closed under boolean operations.

▶ **Theorem 5.** *There is a formula in S-CEL($\mathcal{P}_=^{\mathrm{S}}$) that cannot be expressed in E-CEL($\mathcal{P}_=$).*

An example of a formula that can be defined in S-CEL($\mathcal{P}_=^{\mathrm{S}}$) but cannot be defined in E-CEL($\mathcal{P}_=$) is $\varphi := (R+ ; T+) \texttt{ FILTER } R.id \neq T.id$, where $X.id \neq Y.id$ is defined as $\forall x \in X \forall y \in Y (x(id) \neq y(id))$. Intuitively, an equivalent formula in E-CEL($\mathcal{P}_=$) for φ would need to compare every element in R with every element in T, which requires a quadratic number of comparisons. The proof establishes that the number of comparison in the evaluation of an E-CEL formula is at most linear in the size of the output and, thus, φ cannot be defined by any formula in E-CEL($\mathcal{P}_=$). It is important to note that this result shows the limitations of a CER language based on event variables and what can be gained if set variables are used.

A natural question at this point is whether S-CEL can define every E-CEL formula. For binary predicates (e.g. $x.\texttt{id} = y.\texttt{id}$) the answer is positive, as the following result shows.

▶ **Theorem 6.** *Let $\mathcal{B}$ be any set of event binary predicates closed under complement. Then for every formula $\psi \in$ E-CEL($\mathcal{B}$) there exists a formula $\varphi \in$ S-CEL($\mathcal{B}^{\mathrm{S}}$) such that $\psi \equiv \varphi$.*

It is important to notice that closedness under complement is a mild restriction over $\mathcal{B}$. In particular, if the set $\mathcal{B}$ is closed under boolean operations (as usually every CER query language supports), the condition trivially holds.

Interestingly, it is not true that S-CEL is always more expressive than E-CEL. In particular, there exists an E-CEL formula with ternary predicates that cannot be defined by any S-CEL formula. For the next result, consider the smallest set of event predicates $\mathcal{P}_+$ containing the sum predicate $x = y + z$ that is closed under boolean operations.

▶ **Theorem 7.** *There is a formula in E-CEL($\mathcal{P}_+$) that cannot be expressed in S-CEL($\mathcal{P}_+^{\mathrm{S}}$).*

In the appendix, we show that the formula R AS x ; (S AS y ; T AS z FILTER $(x = y + z)$)+ cannot be defined in S-CEL($\mathcal{P}_+^{\mathrm{S}}$). This formula *injects* the x-variable inside the Kleene closure in order to check that each pair (y, z) sums x. This capability of injecting variables inside Kleene closure cannot be simulated in S-CEL given that in S-CEL a sub-formula cannot filter variables outside its own scope. It is important to recall that this does not occur if binary predicates are used (Theorem 6), which are of common use in CER.

5 On the Expressiveness of Unary Formulas

What is the expressiveness of E-CEL($\mathcal{P}$) or S-CEL($\mathcal{P}$)? To obtain more insight into the the expressive power of the fundamental operators of these languages, we will study this question in the setting where $\mathcal{P}$ is limited to the class $\mathcal{U}$ of unary event predicates. As we showed in Section 4, E-CEL($\mathcal{U}$) and S-CEL($\mathcal{U}^{\mathrm{S}}$) are equally expressive in this setting, suggesting that this is a robust subfragment of CER query languages. In this section, we compare E-CEL and S-CEL with complex event automata (CEA), a computational model proposed in [18] for efficiently evaluating E-CEL with unary event predicates. We show that the so-called $*$-property of CEA captures the expressiveness of E-CEL and S-CEL with unary predicates. Furthermore, we use this property to understand the expressiveness of E-CEL and S-CEL under the extension with new CER operators.

Let $\mathcal{R}$ be a schema and $\mathcal{U}$ be a set of unary event predicates over $\mathcal{R}$. We denote by $\mathcal{U}^+$ the closure of $\mathcal{U} \cup \{\text{tuples}(R) \mid R \in \mathcal{R}\}$ under conjunction. A *complex event automaton* [18] (CEA) over $\mathcal{R}$ and $\mathcal{U}$ is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, $\Delta \subseteq Q \times \mathcal{U}^+ \times \{\circ, \bullet\} \times Q$ is a finite transition relation, and $I, F \subseteq Q$ are the set of initial and final states, respectively. Intuitively, the elements $\{\circ, \bullet\}$ indicate whether or not the element used to take the transition will be part of the output. Given an $\mathcal{R}$-stream $S = t_0 t_1 \ldots$, a run ρ of length n of $\mathcal{A}$ over S is a sequence of transitions $\rho : q_0 \xrightarrow{P_0/m_0} q_1 \xrightarrow{P_1/m_1} \cdots \xrightarrow{P_n/m_n} q_{n+1}$ such that $q_0 \in I$, $t_i \in P_i$ and $(q_i, P_i, m_i, q_{i+1}) \in \Delta$ for every $i \leq n$. ρ is *accepting* if $q_{n+1} \in F$. $\text{Run}_n(\mathcal{A}, S)$ denotes the set of accepting runs of $\mathcal{A}$ over S of length n. Further, we define the complex event $C \subseteq 2^{\mathbb{N}}$ induced by ρ as

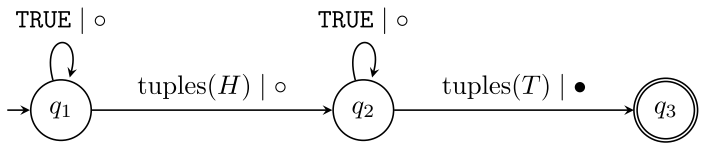

Figure 2 A complex event automaton that has no equivalent formula in S-CEL.

$C_\rho = \{i \in [0,n] \mid m_i = \bullet\}$. Given a stream S and $n \in \mathbb{N}$, we define the set of complex events of $\mathcal{A}$ over S at position n as $[\![\mathcal{A}]\!]_n(S) = \{C_\rho \mid \rho \in \text{Run}_n(\mathcal{A}, S)\}$.

In [18], it was shown that for every formula $\varphi \in$ E-CEL($\mathcal{U}$) there exists an equivalent CEA $\mathcal{A}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]_n(S)$ for every stream S and position n. By Theorem 4, it follows that for every formula $\varphi \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$) there is an equivalent CEA $\mathcal{A}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]_n(S)$ for every stream S and position n. It is then natural to ask whether the converse also holds, namely, if every CEA $\mathcal{A}$ over $\mathcal{U}$ has an equivalent formula in S-CEL($\mathcal{U}^{\mathrm{S}}$) (and thus in E-CEL($\mathcal{U}$)). Here, however, the answer is negative because CEA can make decisions based on tuples that are not part of the output complex event, while formulas cannot. Consider for example the CEA of Figure 2. This automaton will output complex events of the form $C = \{i\}$, provided that $S[i]$ is of type T and there is a previous position $j < i$ such that $S[j]$ is of type H. It is straightforward to prove that this cannot be achieved by S-CEL formulas because such a formula would either not check that the H events occurs, or include the position j of H in C – which the automaton does not.

In order to capture the exact expressiveness of E-CEL or S-CEL formulas with unary predicates, we restrict CEA to a new semantics called the *$*$-semantics*. Formally, let $\mathcal{A} = (Q, \Delta, I, F)$ be a complex event automaton and $S = t_1, t_2, \ldots$ be a stream. A $*$-run ρ^* of $\mathcal{A}$ over S ending at n is a sequence of transitions: $\rho^* : (q_0, 0) \xrightarrow{P_1/\bullet} (q_1, i_1) \xrightarrow{P_2/\bullet} \cdots \xrightarrow{P_k/\bullet} (q_k, i_k)$ such that $q_0 \in I$, $0 < i_1 < \ldots < i_k = n$ and, for every $j \geq 1$, $(q_{j-1}, P_j, \bullet, q_j) \in \Delta$ and $S[i_j] \in P_j$. We say that ρ^* is an accepting $*$-run if $q_k \in F$. Furthermore, we denote by $C_\rho \subseteq 2^{\mathbb{N}}$ the complex event induced by ρ^* as $C_{\rho^*} = \{i_j \mid j \leq k\}$. The set of all complex events generated by $\mathcal{A}$ over S under the $*$-semantics is defined as: $[\![\mathcal{A}]\!]^*_n(S) = \{C_{\rho^*} \mid \rho^*$ is an accepting $*$-run of $\mathcal{A}$ over S ending at $n\}$. Notice that under this semantics, the automaton no longer has the ability to verify a tuple without marking it but it is allowed to skip an arbitrary number of tuples between two marking transitions.

We can now effectively capture the expressiveness of unary formulas as follows.

▶ **Theorem 8.** *For every set $\mathcal{U}$ of unary event predicates, S-CEL($\mathcal{U}^{\mathrm{S}}$) has the same expressive power as CEA($\mathcal{U}$) under the $*$-semantics, namely, for every formula φ in S-CEL($\mathcal{U}^{\mathrm{S}}$), there exists a CEA $\mathcal{A}$ over $\mathcal{U}$ such that $[\![\varphi]\!]_n(S) = [\![\mathcal{A}]\!]^*_n(S)$ for every S and n, and vice versa.*

For every stream S and complex event C, let $S[C\rangle$ refer to the subsequence of S induced by C. An interesting property of the $*$-semantics is that, for every CEA $\mathcal{A}$, stream S, and complex event $C \in [\![\mathcal{A}]\!]^*(S)$, we can arbitrarily modify, add and remove tuples in S that are not mentioned in $S[C\rangle$, and the original tuples in $S[C\rangle$ would still form a complex event of $\mathcal{A}$ over the new stream. To formalize this, we need some additional definitions. A *stream-function* f is a function f : streams($\mathcal{R}$) $\to 2^{\mathbf{C}}$, where streams($\mathcal{R}$) is the set of all $\mathcal{R}$-streams and $\mathbf{C}$ is the set of all complex events. Although f can be any function that returns a set of complex events on input streams, we are interested in the processing-functions f that can be described either by a S-CEL formula φ (i.e. $f = [\![\varphi]\!]$) or by a CEA $\mathcal{A}$ (i.e. $f = [\![\mathcal{A}]\!]$). Let S_1, S_2 be two streams and C_1, C_2 be two complex events. We say that S_1 and C_1 are *$*$-related* with S_2 and C_2, written as $(S_1, C_1) =_* (S_2, C_2)$, if $S_1[C_1\rangle = S_2[C_2\rangle$.

Consider now a stream-function f. We say that f has the *$*$-property* if, for every stream S and complex event $C \in f(S)$, it holds that $C' \in f(S')$ for every S' and C' such that $(S, C) =_* (S', C')$. A way to understand the $*$-property is to see S' as the result of fixing the tuples in S that are part of $S[C]$ and adding or removing tuples arbitrarily, and defining C' to be the complex event that has the same original tuples of C. The following proposition states the relation that exists between the $*$-property and the $*$-semantics over CEA.

▶ **Proposition 9.** *If the stream-function defined by a CEA $\mathcal{A}$ has the $*$-property, then there exists a CEA $\mathcal{A}'$ such that $[\![\mathcal{A}]\!]_n(S) = [\![\mathcal{A}']\!]^*_n(S)$ for every S and n.*

By combining Theorem 8 and Proposition 9 we get the following result.

▶ **Corollary 10.** *Let f be a stream-function. Then f can be defined by a CEA over $\mathcal{U}$ and has the $*$-property iff there exists a formula φ in S-CEL($\mathcal{U}^{\mathrm{S}}$) such that $f = [\![\varphi]\!]$.*

With the previous corollary we have captured the exact expressiveness of E-CEL($\mathcal{U}$) and S-CEL($\mathcal{U}^{\mathrm{S}}$) based on a restricted subclass of CEA. Interestingly, we can use this characterization to show that other operators for CER that have been proposed in the literature [12] can be captured by S-CEL($\mathcal{U}^{\mathrm{S}}$). Some languages include additional useful operators like `AND`, `ALL` and `UNLESS`, which have the following semantics in S-CEL. Given a complex event C, a stream S, a valuation μ, and $i, j \in \mathbb{N}$:

- $C \in [\![\rho_1 \texttt{ AND } \rho_2]\!](S, i, j, \mu)$ iff $C \in [\![\rho_1]\!](S, i, j, \mu) \cap [\![\rho_2]\!](S, i, j, \mu)$.
- $C \in [\![\rho_1 \texttt{ ALL } \rho_2]\!](S, i, j, \mu)$ if and only if there are $i_1, i_2, j_1, j_2 \in \mathbb{N}$, complex events C_1, C_2, and valuations μ_1, μ_2 such that $C_k \in [\![\rho_k]\!](S, i_k, j_k, \mu_k)$, $C = C_1 \cup C_2$, $\mu = \mu_1 \cup \mu_2$, $i = \min\{i_1, i_2\}$ and $j = \max\{j_1, j_2\}$.
- $C \in [\![\rho_1 \texttt{ UNLESS } \rho_2]\!](S, i, j, \mu)$ iff $C \in [\![\rho_1]\!](S, i, j, \mu)$ and, for every complex event C', valuation μ', and $i', j' \in \mathbb{N}$ such that $i \leq i' \leq j' \leq j$, it holds that $C' \notin [\![\rho_2]\!](S, i', j', \mu')$.

The `AND` operator selects those matches produced by both formulas. Although this is natural for sets, it is restrictive for capturing events. On the contrary, `ALL` is more flexible and allows to combine complex events. In this sense, `ALL` is similar to sequencing but allows the complex events to occur at any point in time, even overlapping or intersecting. For example, suppose that we want to capture a temperature below 0 degrees and a humidity over 60% that can occur in any order. This can be written as $(T \texttt{ ALL } H) \texttt{ FILTER } (T.value < 0 \wedge H.value \geq 60)$. The motivation for introducing `UNLESS` in CER languages is to have some sort of negation [12]. It is important to mention that the *negated* formula (the right-hand side) is restricted to complex events between the start and end of complex events for the formula in the left-hand side. This is motivated by the fact that a complex event should not depend on objects that are distant in the stream. For example, consider that we want to see a drastic increase in temperature, i.e., a sequence of a low temperature (less than 20 degrees) followed by a high temperature (more than 40 degrees), where no other temperatures occur in between. This can be expressed by the following pattern with the `UNLESS` operator:

$$\big[(T \texttt{ IN } TF\,;\, T \texttt{ IN } TL) \texttt{ FILTER } (TF.value < 20 \wedge TL.value > 40)\big]$$
$$\texttt{UNLESS } [T \texttt{ FILTER } (T.value >= 20 \wedge T.value <= 40)]$$

Interestingly, from a language design point of view, the operators `AND` and `ALL` are redundant in the sense that `AND` and `ALL` do not add expressive power in the unary case. Indeed, `AND` and `ALL` can be defined by CEA and both satisfy the $*$-property.

▶ **Corollary 11.** *Let $\mathcal{U}$ be a set of unary event predicates. For every expression φ of the form φ_1 `OP` φ_2, with `OP` $\in \{$`AND`, `ALL`$\}$ and φ_i in S-CEL($\mathcal{U}^{\mathrm{S}}$), there is a S-CEL($\mathcal{U}^{\mathrm{S}}$) formula φ' such that $[\![\varphi]\!]_n(S) = [\![\varphi']\!]_n(S)$ for every S and n.*

In contrast, the `UNLESS` operator can be defined by CEA but one can show that there are formulas mentioning `UNLESS` that do not satisfy the $*$-property. Then, by Corollary 10, `UNLESS` is not expressible in S-CEL($\mathcal{U}^{\mathrm{S}}$) with $\mathcal{U}$ unary event predicates. This shows that `UNLESS` adds expressibility to unary S-CEL formulas while remaining executable by CEA.

6 Capturing the Expressive Power of Complex Event Automata

As discussed in Section 5, given a set $\mathcal{U}$ of unary event predicates, S-CEL($\mathcal{U}^{\mathrm{S}}$) captures the class of CEA over $\mathcal{U}$ that have the $*$-property (Corollary 10). However, in [18] it was shown that all CEA can be evaluated efficiently, and not only those satisfying the $*$-property. It makes sense then to study the origin of this lack of expressive power and extend the language to precisely capture the expressiveness of the automata model.

6.1 Expressibility of CEA and Unary S-CEL

By looking at the characterization of S-CEL in terms of the $*$-property, one can easily distinguish three shortcomings of S-CEL. First, every event that is relevant for capturing a complex event must be part of the output. Although this might be a desired property in some cases, it disallows projecting over a subset of the relevant events. This limitation is explained by the $*$-property, and suggests that to capture CEA we need an operator that allows to remove or, in other words, project events that must appear in the stream but are irrelevant for the output. Although projection is one of the main operators in relational databases, it is rarely used in the context of CER, possibly because of the difficulties encountered when trying to define a consistent semantics that combines projection with operators like Kleene closure. Interestingly, we show below that by using set variables it is straightforward to introduce a simple projection operator in S-CEL.

The second shortcoming of S-CEL is that it cannot express contiguous sequences. The sequencing operators (; and +) allow for arbitrary *irrelevant* events in between. While this is a typical requirement in CER, a user could want to capture contiguous events, which has been considered in some CER language before [24] as a *selection operator* that keeps contiguous sequences of events in the output (see Section 6.2 for further discussion). Given that this can be naturally achieved by CEA and has been previously proposed in the literature, it is reasonable to include some operators that allow to declare contiguous sequence of events.

A final feature that is clearly supported by CEA but not by S-CEL is specifying that a complex event starts at the beginning of the stream. This feature is not particularly interesting in CER, but we include it as a new operator with the simple objective of capturing the computational model. Actually, this operator is intensively used in the context of regular expression programing where an expression of the form "$^\wedge R$" marks that R must be evaluated starting from the beginning of the document. Therefore, it is not at all unusual in query languages to include an operator that recognizes events from the beginning of the stream.

Given the discussion above, we propose to extend S-CEL with the following operators:

$$\varphi \quad := \quad \varphi : \varphi \;\mid\; \varphi\oplus \;\mid\; \pi_L(\varphi) \mid\; \texttt{START}(\varphi)$$

where $L \subseteq \mathbf{L}$. Recall that for a valuation μ, $\mathrm{supp}(\mu)$ is defined as $\mathrm{supp}(\mu) = \bigcup_{A \in \mathbf{L}} \mu(A)$. Given a formula φ of one of the forms above, a complex event C, a stream S, a valuation μ, and positions i, j, we say that $C \in [\![\varphi]\!](S, i, j, \mu)$ if one of the following conditions holds:

- $\varphi = \rho_1 : \rho_2$ and there exists two non-empty complex events C_1 and C_2 and valuations μ_1 and μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in [\![\rho_1]\!](S, i, \max(C_1), \mu_1)$, $C_2 \in [\![\rho_2]\!](S, \min(C_2), j, \mu_2)$ and $\max(C_1) = \min(C_2) - 1$.
- $\varphi = \rho\oplus$ and either $C \in [\![\rho]\!](S, i, j, \mu)$ or $C \in [\![\rho : \rho\oplus]\!](S, i, j, \mu)$.
- $\varphi = \pi_L(\rho)$, $C = \mathrm{supp}(\mu)$ and there is $C' \in [\![\rho]\!](S, i, j, \mu')$ for some valuation μ' such that $\mu(A) = \mu'(A)$ if $A \in L$ and $\mu(A) = \emptyset$ otherwise.
- $\varphi = \ \mathtt{START}(\rho)$, $C \in [\![\varphi']\!](S, i, j, \mu)$, and $\min(C) = i$.

To denote the extension of S-CEL with a set of operators $\mathcal{O}$ we write S-CEL$\,\cup\mathcal{O}$. For readability, we use the special notation S-CEL+ to denote S-CEL$\,\cup\{\,:\,, \oplus, \pi, \mathtt{START}\}$.

The idea behind : and $\oplus$ is to simulate ; and +, respectively, but imposing that *irrelevant* events cannot occur in between. This allows us to recognize, for example, the occurrence of an event of type R immediately after an event of type T ($\varphi = R : T$), or an unbounded series of consecutive events of type R ($\varphi = R\oplus$). Note, however, that the operator $\oplus$ does not impose that intermediate events are contiguous. For example the formula $(R; S)\oplus$ imposes that the last event S of one iteration occurs right before the first event R of the next iteration, but in one iteration the R event and the S event do not need to occur contiguously.

▶ **Example 12.** Following the schema of our running example, suppose that we want to detect a period of temperatures below 0° and humidities below 40%, followed by a sudden increase of humidity (above 45%). Naturally, we do not expect to skip *irrelevant* temperatures or humidities, as this would defy the purpose of the pattern. Assuming that we are only interested in retrieving the humidity measurements, this pattern would be written as follows:

$$\pi_H[((H \ \mathtt{IN}\ X) \ \mathtt{OR}\ T)\oplus : (H \ \mathtt{IN}\ Y) \ \mathtt{FILTER}\ (X.value < 40 \wedge T.value < 0 \wedge Y.value > 45)].$$

Having defined the previous operators, we proceed to show that for every set $\mathcal{U}$ of unary predicates, S-CEL+($\mathcal{U}^{\mathrm{S}}$) captures the full expressive power of CEA over $\mathcal{U}$. To this end, we say that a formula φ in S-CEL+($\mathcal{U}^{\mathrm{S}}$) is equivalent to a CEA $\mathcal{A}$ over $\mathcal{U}$ (denoted by $\varphi \equiv \mathcal{A}$) if for every stream S and $n \in \mathbb{N}$ it is the case that $[\![\mathcal{A}]\!]_n(S) = [\![\varphi]\!]_n(S)$.

▶ **Theorem 13.** *Let $\mathcal{U}$ be a set of unary event predicates. For every CEA $\mathcal{A}$ over $\mathcal{U}$, there is a formula $\varphi \in$ S-CEL+($\mathcal{U}^{\mathrm{S}}$) such that $\varphi \equiv \mathcal{A}$. Conversely, for every formula $\varphi \in$ S-CEL+($\mathcal{U}^{\mathrm{S}}$) there exists a CEA $\mathcal{A}$ over $\mathcal{U}$ such that $\varphi \equiv \mathcal{A}$.*

This result is particularly relevant because, as shown in [18], for every stream S and CEA $\mathcal{A}$, we can evaluate $\mathcal{A}$ by consuming the stream S using constant time to process every new event, and after consuming the n^{th} event of S the set $[\![\mathcal{A}]\!]_n(S)$ is enumerated with *constant delay*. Although the constants here are measured under data complexity and might depend exponentially on the size of the automaton, these are useful efficiency guarantees for CER in practice, and therefore extending S-CEL to a language that precisely captures the class of CEA gives more expressive power while maintaining these efficiency guarantees.

6.2 Strict Sequencing versus Strict Selection

For recognizing events that occur contiguously we introduced the strict-sequencing operators (i.e. : and $\oplus$) that locally check this condition. These operators are the natural extension of ; and +, and they resemble the standard operators of concatenation and Kleene star from regular expressions. However, to the best of our knowledge strict-sequencing has not been proposed before in the context of CER, possibly because adding this feature to a language might complicate the semantics, specially when combined with other non-strict operators. To avoid this interaction, the strict-contiguity selection (or strict-selection) has been previously introduced in [24] by means of a unary predicate that basically forces a complex event C to capture a contiguous set of events. Formally, for any formula φ in S-CEL let $\mathtt{STRICT}(\varphi)$ be the syntax for the strict-selection operator previously mentioned. Given a stream S, a valuation μ, and two position $i, j \in \mathbb{N}$, we say that $C \in [\![\mathtt{STRICT}(\varphi)]\!](S, i, j, \mu)$ if $C \in [\![\varphi]\!](S, i, j, \mu)$ and C is an interval (i.e. there are no $i, k \in C$ and $j \notin C$ s.t. $i < j < k$).

A reasonable question is whether the same expressiveness results of Theorem 13 could be obtained with STRICT. We answer this by giving evidence that our decision of including strict-sequencing operators instead of strict-selection was correct. We show that strict-sequencing and strict-selection coincide if we restrict our comparison to unary predicates. Surprisingly, if we move to binary predicates, strict-selection is strictly less expressive than strict-sequencing.

At a first sight, the strict-sequencing operators and the strict-selection predicates seems equally expressive since both allows to force contiguity between pair of events. At least, this intuition holds whenever we restrict to unary predicates.

▶ **Proposition 14.** *Let $\mathcal{U}$ be a set of unary set predicates. For every φ in S-CEL$\cup\{:, \oplus\}(\mathcal{U})$, there exists a formula ψ in S-CEL$\cup\{\mathtt{STRICT}\}(\mathcal{U})$ such that $\varphi \equiv \psi$, and vice-versa.*

The connection between both operators change if we move to predicates of higher arity. Note, however, that STRICT can always be simulated by the sequencing operators $:$ and $\oplus$.

▶ **Proposition 15.** *Let $\mathcal{P}$ be a set of set predicates. Given a formula $\varphi \in$ S-CEL$\cup\{\mathtt{STRICT}\}(\mathcal{P})$ there exists $\psi \in$ S-CEL$\cup\{:, \oplus\}(\mathcal{P})$ such that $\varphi \equiv \psi$.*

To explain our decision of including the operators $:$ and $\oplus$ instead of STRICT, we study the opposite direction. First, it is not hard to see that the operator $:$ can indeed be simulated by means of the operator STRICT. Actually, for any formula $\varphi_1 : \varphi_2$ we can isolate the rightmost and leftmost event definition of φ_1 and φ_2 respectively, change $:$ by ; and surround it by a STRICT operator. Now, if we include the operator $\oplus$, the situation becomes more complex. In particular, for binary predicates, STRICT is not capable of simulating the $\oplus$-operator.

▶ **Theorem 16.** *For any set $\mathcal{P}$ of set predicates and for any formula $\varphi \in$ S-CEL$\cup\{:\}(\mathcal{P})$ there is a formula $\psi \in$ S-CEL$\cup\{\mathtt{STRICT}\}(\mathcal{P})$ such that $\varphi \equiv \psi$. In contrast, there exists a set $\mathcal{P}$ containing a single binary set predicate and a formula $\varphi \in$ S-CEL$\cup\{\oplus\}(\mathcal{P})$ that is not equivalent to any formula in S-CEL$\cup\{\mathtt{STRICT}\}(\mathcal{P})$.*

This last theorem concludes our discussion on the operators for contiguity, and allows us to argue that including the operators $:$ and $\oplus$ is better than including the unary operator STRICT. It is worth noting that the proof of Theorem 16 is a non-trivial result that requires a version of the pumping lemma for CEA; the proof can be found in the Appendix.

7 Discussion and future work

There are several future research directions regarding the relation between CER languages, logics, and streaming evaluation. For example, one relevant problem is to understand the connection between S-CEL and monadic second-order logic (MSO). For unary filters, we conjecture that S-CEL+ has the same expressive power as MSO over unary filters. Another natural question is to compare the expressiveness of S-CEL+ and MSO extended with binary predicates. Furthermore, a more fundamental question is what fragments of S-CEL or MSO (with binary predicates) can be evaluated with strong guarantees like constant-delay enumeration. We believe that understanding the relation between S-CEL, formal logics (e.g. MSO), and constant delay algorithms is fundamental for the design of CER languages and the implementation of CER systems.

References

1 Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 147–160, 2008.

2 Alfred V. Aho. Algorithms for finding patterns in strings. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 255–300. North Holland, 1990.

3 Alexander Artikis, Alessandro Margara, Martín Ugarte, Stijn Vansummeren, and Matthias Weidlich. Complex event recognition languages: Tutorial. In *International Conference on Distributed and Event-based Systems, DEBS 2017,*, pages 7–10, 2017.

4 Alexander Artikis, Marek J. Sergot, and Georgios Paliouras. An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908, 2015.

5 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.*, 14(6):1007–1018, 2003.

6 Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015. URL: http://sites.computer.org/debull/A15dec/p28.pdf.

7 Benjamin Carle and Paliath Narendran. On extended regular expressions. In *LATA 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 279–289, 2009.

8 Badrish Chandramouli, Jonathan Goldstein, Mike Barnett, Robert DeLine, John C. Platt, James F. Terwilliger, and John Wernsing. Trill: A high-performance incremental query processor for diverse analytics. *PVLDB*, 8(4):401–412, 2014. URL: http://www.vldb.org/pvldb/vol8/p401-chandramouli.pdf, doi:10.14778/2735496.2735503.

9 Charles D. Cranor, Yuan Gao, Theodore Johnson, Vladislav Shkapenyuk, and Oliver Spatscheck. Gigascope: high performance network monitoring with an SQL interface. In *SIGMOD*, page 623, 2002.

10 Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD*, pages 647–651, 2003.

11 Gianpaolo Cugola and Alessandro Margara. Complex event processing with t-rex. *The Journal of Systems and Software*, 2012.

12 Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 2012.

13 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. A general algebra and implementation for monitoring event streams. Technical report, Cornell University, 2005.

14 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

15 Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *CIDR 2007*, pages 412–422, 2007.

16 Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Sase+: An agile language for kleene closure over event streams. Technical report.

17 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12:1–12:51, 2015. URL: http://doi.acm.org/10.1145/2699442, doi:10.1145/2699442.

18 Alejandro Grez, Cristian Riveros, and Martin Ugarte. A formal framework for complex event processing. In *ICDT*, 2019.

19 Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning temporal regularity in video sequences. In *2016 Conference on Computer Vision and Pattern Recognition*, pages 733–742, 2016.

20 Martin Hirzel, Guillaume Baudart, Angela Bonifati, Emanuele Della Valle, Sherif Sakr, and Akrivi Vlachou. Stream processing languages in the big data era. *SIGMOD Record*, 47(2):29–40, 2018. URL: https://doi.org/10.1145/3299887.3299892, doi:10.1145/3299887.3299892.

21 Ilya Kolchinsky, Izchak Sharfman, and Assaf Schuster. Lazy evaluation methods for detecting complex events. In *International Conference on Distributed Event-Based Systems, DEBS '15*, pages 34–45, 2015.

22 Leonid Libkin. *Elements of finite model theory*. Springer Science & Business Media, 2013.

23 Walker M. White, Mirek Riedewald, Johannes Gehrke, and Alan J. Demers. What is "next" in event processing? In *PODS*, pages 263–272, 2007.

24 Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD*, 2006.

25 Haopeng Zhang, Yanlei Diao, and Neil Immerman. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*, 2014.

A Proofs of Section 4

A.1 Proof of Theorem 4

To prove the theorem we use the fact that, when dealing with event unary predicates, one can always rewrite the formulas so that all predicates are applied at the lower level, directly on the assignments. In E-CEL($\mathcal{U}$) formulas, this notion is defined on [18] as *locally-parametrized normal form*, or LP normal form. The syntax of formulas in LP normal form is restricted to the following grammar:

$$\varphi \ := \ R \ \texttt{AS} \ x \ \mid \ R \ \texttt{AS} \ x \ \texttt{FILTER} \ P_1(x) \wedge \ldots \wedge P_k(x) \ \mid \ \varphi \ \texttt{OR} \ \varphi \ \mid \ \varphi \,;\, \varphi \ \mid \ \varphi+$$

Where R is a relation, x is a variable and $P_1, \ldots, P_k$ are predicates of $\mathcal{U}$. To simplify the presentation of the proof, when writing a conjunction of predicates on the filters, it is short for a series of nested filters, were each one is one of the predicates. In [18], they give a construction that, for every E-CEL($\mathcal{U}$) formula, defines an equivalent formula in LP normal form.

For S-CEL($\mathcal{U}^{\mathrm{S}}$) formulas, we show now that one can rewrite them to get a similar structure by pushing down every predicate. This is a rather predictable property, since every predicate $P \in \mathcal{U}^{\mathrm{S}}$ is a universally quantified extension of one in $\mathcal{U}$, thus if a set A satisfies P, then every $A' \subseteq A$ also satisfies P. Following this idea we show that, for every φ $\texttt{FILTER}$ $P(A) \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$), if φ is not an atomic formula (i.e. $\varphi \neq R$ and $\varphi \neq R$ $\texttt{FILTER}$ $P_1(R) \wedge \ldots \wedge P_k(R)$), then $P(A)$ can be pushed one level deeper in φ. We consider the possible cases of φ:

- If $\varphi = \varphi_1$ $\texttt{OP}$ φ_2, with $\texttt{OP} \in \{$ $\texttt{OR}$, ; $\}$, then

$$\varphi \ \texttt{FILTER} \ P(A) \ \equiv \ \varphi_1 \ \texttt{FILTER} \ P(A) \ \texttt{OP} \ \varphi_2 \ \texttt{FILTER} \ P(A).$$

- If $\varphi = \varphi_1+$, then φ $\texttt{FILTER}$ $P(A) \equiv (\varphi_1$ $\texttt{FILTER}$ $P(A))+$.
- If $\varphi = \varphi_1$ $\texttt{IN}$ B:
  - if $B \neq A$, then φ $\texttt{FILTER}$ $P(A) \equiv (\varphi_1$ $\texttt{FILTER}$ $P(A))$ $\texttt{IN}$ B,
  - if $B = A$, then φ $\texttt{FILTER}$ $P(A) \equiv (\varphi_1$ $\texttt{FILTER}$ $P(A_1) \wedge P(A_2) \wedge \cdots \wedge P(A_n))$ $\texttt{IN}$ A, where $A_1, \ldots, A_n$ are the assigned labels in φ_1.

The correctness of these equivalences follows straightforward from the definition of the semantics. Then, by using these equivalences one can push all the predicates down, and the syntax of the resulting formula is of the form:

$$\varphi \ := \ R \ \mid \ R \ \texttt{FILTER} \ P_1(R) \wedge \ldots \wedge P_k(R) \ \mid \ \varphi \ \texttt{OR} \ \varphi \ \mid \ \varphi \,;\, \varphi \ \mid \ \varphi+$$

where R is a relation and $P_1, \ldots, P_k \in \mathcal{U}^{\mathrm{S}}$. Notice that we dropped the $\texttt{IN}$ operator. This is because all filters are applied on the assignments, therefore the labels do not change anything in the results. We say a S-CEL formula with unary predicates is in LP normal form if it has this syntax.

Now the construction to go between E-CEL($\mathcal{U}$) and S-CEL($\mathcal{U}^{\mathrm{S}}$) is straightforward. First, if we have a formula $\varphi \in$ E-CEL($\mathcal{U}$), we can assume w.l.o.g. that it is in LP normal form, and then replace every R $\texttt{AS}$ x $\texttt{FILTER}$ $P(x)$ with R $\texttt{FILTER}$ $P^{\mathrm{S}}(R)$, where P^{S} is the set extension of P, to get a formula in S-CEL($\mathcal{U}^{\mathrm{S}}$). Similarly for the other direction, if we have a formula $\psi \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$), we can assume w.l.o.g. it is in LP normal form, and then replace every R $\texttt{FILTER}$ $P^{\mathrm{S}}(R)$ with R $\texttt{AS}$ x $\texttt{FILTER}$ $P(x)$, where x is a new variable.

Let $\varphi \in$ E-CEL($\mathcal{U}$) in LP normal form, and let $\psi \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$) the resulting formula (also in LP normal form) from the construction above. For every stream S, complex event C and positions $i < j$ we prove that $C \in [\![\varphi]\!](S, i, j, \nu)$ for some ν iff $C \in [\![\psi]\!](S, i, j, \mu)$ for some μ. In particular, this implies that $[\![\varphi]\!](S) = [\![\psi]\!](S)$. We prove by doing induction over the structure of the formula. In the following, we prove only for one direction, i.e. $C \in [\![\varphi]\!](S, i, j, \nu)$ implies $C \in [\![\psi]\!](S, i, j, \mu)$; the opposite direction holds directly by following the arguments in reversed order.

The base case is when $\varphi = R$ `AS` x `FILTER` $P(x)$ and $\psi = R$ `FILTER` $P^{\mathrm{S}}(R)$. Consider $C \in [\![R \texttt{ AS } x \texttt{ FILTER } P(x)]\!](S, i, j, \nu)$ for some ν. Then, by definition we have: $C = \{\nu(x)\}$, type($S[\nu(x)]) = R$, $i \leq \nu(x) = j$ and $S[\nu(x)] \in P$. Now consider the set valuation μ having $\mu(R) = \{j\}$ and $\mu(A) = \emptyset$ for all $A \neq R$. It is clear then that $C = \mu(R) = j$, type($S[j] = R$ and $S[\mu(R)] \in P^{\mathrm{S}}$, satisfying the conditions for $C \in [\![\psi]\!](S, i, j, \mu)$. Now for the inductive step consider the following cases of φ.

- If $\varphi = \varphi_1$ `OR` φ_2, then either $C \in [\![\varphi_1]\!](S, i, j, \nu)$ or $C \in [\![\varphi_2]\!](S, i, j, \nu)$; w.l.o.g. assume the former. From the construction, ψ has the form ψ_1 `OR` ψ_2 and, by induction hypothesis, $C \in [\![\psi_1]\!](S, i, j, \mu)$ for some μ, thus $C \in [\![\psi]\!](S, i, j, \mu)$.
- If $\varphi = \varphi_1 \,;\, \varphi_2$, then by definition there exists k and complex events C_1, C_2 such that $C = C_1 \cdot C_2$, $C_1 \in [\![\varphi_1]\!](S, i, k, \nu)$ and $C_2 \in [\![\varphi_2]\!](S, k+1, j, \nu)$. From the construction, ψ has the form $\psi_1 \,;\, \psi_2$ and, y induction hypothesis, $C_1 \in [\![\psi_1]\!](S, i, k, \mu_1)$ and $C_2 \in [\![\psi_2]\!](S, k{+}1, j, \mu_2)$ for some μ_1, μ_2. Then, we satisfy all the conditions for $C \in [\![\psi]\!](S, i, j, \mu)$ if we define $\mu = \mu_1 \cup \mu_2$.
- If $\varphi = \varphi'+$, then by the construction, ψ has the form $\psi'+$. Recall the semantics of $C \in [\![\varphi'+]\!]$: $C \in \bigcup_{l=1}^{\infty} [\![\rho[l]]\!](S, i, j, \nu)$, meaning that there exists a valuation ν' such that either $C \in [\![\varphi']\!](S, i, j, \nu[\nu'/U])$ if $l = 1$ or $C \in [\![\varphi' \,;\, \varphi'[l-1]]\!](S, i, j, \nu[\nu'/U])$ otherwise, where $U = \mathrm{vdef}_+(\varphi')$.

  We prove that $C \in [\![\psi'+]\!](S, i, j, \mu)$ for some μ by doing a second induction, this time over the number of iterations l over the definition of $+$. The base case is when the non-recursive definition is used, i.e., $C \in [\![\varphi']\!](S, i, j, \nu[\nu'/U])$, for some ν'. Then, by the first induction hypothesis $C \in [\![\psi']\!](S, i, j, \mu)$ for some μ, and therefore $C \in [\![\psi'+]\!](S, i, j, \mu)$.

  For the inductive step, consider $C \in [\![\varphi'[l]]\!](S, i, j, \nu)$. By the definition of $+$, $C \in [\![\varphi' \,;\, \varphi'[l-1]]\!](S, i, j, \nu[\nu'/U])$, and by definition of ; there exists k and C_1, C_2 such that $C = C_1 \cdot C_2$, $C_1 \in [\![\varphi']\!](S, i, k, \nu[\nu'/U])$ and $C_2 \in [\![\varphi'+]\!](S, k+1, j, \nu[\nu'/U])$. From the first induction hypothesis, $C_1 \in [\![\psi']\!](S, i, k, \mu_1)$ for some μ_1; from the second induction hypothesis, $C_2 \in [\![\psi'+]\!](S, k+1, j, \mu_2)$ for some μ_2 (because it used $l-1$ number of iterations). Then, by defining $\mu = \mu_1 \cup \mu_2$ we get $C \in [\![\psi' \,;\, \psi'+]\!](S, i, j, \mu)$, therefore $C \in [\![\psi'+]\!](S, i, j, \mu)$.

Therefore, the two-way construction allows us to give for each $\varphi \in$ E-CEL($\mathcal{U}$) an equivalent formula $\psi \in$ S-CEL($\mathcal{U}^{\mathrm{S}}$), and vice versa, proving that both logics are equally expressive.

A.2 Proof of Theorem 5

Here we prove that the formula $\varphi = (R+ \,;\, T+)$ `FILTER` $R \neq T)$ in S-CEL($\mathcal{P}_{=}^{\mathrm{S}}$) does not have an equivalent formula in E-CEL($\mathcal{P}_{=}$) (for simplification, we talk about predicates between elements, rather than between element attributes). Intuitively, an equivalent E-CEL($\mathcal{P}_{=}$) formula for φ would need to compare every element in R with every element in T (i.e. a quadratic number of comparisons). In the sequel we show that the number of comparisons in the evaluation of an E-CEL($\mathcal{P}_{=}$) formula is at most linear in the size of the output, and therefore, φ cannot be defined by E-CEL($\mathcal{P}_{=}$).

To formalize the notion of the comparisons associated to an output, we extend the semantics of E-CEL in the following way. First, we define a comparing set O as a set of tuples, where the first element is a predicate and the followings are positions. For example, a valid element of O is $(=, 1, 3)$, which represents that the events at positions 1 and 3 were compared with equality, i.e. $S[1] = S[3]$. Strictly speaking, we should also add the information about the attributes that were being compared, but we leave that out to keep notation simple. Now, given a formula ψ in E-CEL($\mathcal{P}$) a complex event C, a comparing set O, a stream S and positions i, j, we say that $(C, O) \in [\![\psi]\!](S, i, j, \nu)$ if:

- $\psi = R$ `AS` x, $C = \{\nu(x)\}$, $\text{type}(S[\nu(x)]) = R$, $i \leq \nu(x) = j$ and $O = \emptyset$.
- $\psi = \rho$ `FILTER` $P(x_1, \ldots, x_n)$, there exists some comparing set O' such that $(C, O') \in [\![\rho]\!](S, i, j, \nu)$, $(S[\nu(x_1)], \ldots, S[\nu(x_n)]) \in P$ and $O = O' \cup \{(P, \nu(x_1), \ldots, \nu(x_n))\}$.
- $\psi = \rho_1$ `OR` ρ_2 and $(C, O) \in [\![\rho_1]\!](S, i, j, \nu)$ or $(C, O) \in [\![\rho_2]\!](S, i, j, \nu))$.
- $\psi = \rho_1 \,;\, \rho_2$ and there exist $k \in \mathbb{N}$, complex events C_1 and C_2 and comparing sets O_1 and O_2 such that $C = C_1 \cdot C_2$, $O = O_1 \cup O_2$, $(C_1, O_1) \in [\![\rho_1]\!](S, i, k, \nu)$ and $(C_2, O_2) \in [\![\rho_2]\!](S, k+1, j, \nu)$.
- $\psi = \rho+$ and $(C, O) \in \bigcup_{k=1}^{\infty} [\![\rho[k]]\!](S, i, j, \nu)$, where $(C, O) \in [\![\rho[k]]\!](S, i, j, \nu)$ if there exists a valuation ν' such that either $(C, O) \in [\![\rho]\!](S, i, j, \nu[\nu'/U])$ and $k = 1$ or $(C, O) \in [\![\rho \,;\, \rho[k-1]]\!](S, i, j, \nu[\nu'/U])$ otherwise, where $U = \text{vdef}_+(\rho)$.

Notice that we only extended the previous semantics of E-CEL adding this new notion of comparing set. Therefore, it is not hard to see that $(C, O) \in [\![\psi]\!](S, i, j, \nu)$ implies $C \in [\![\psi]\!](S, i, j, \nu)$ and, conversely, that $C \in [\![\psi]\!](S, i, j, \nu)$ implies there is some O such that $(C, O) \in [\![\psi]\!](S, i, j, \nu)$.

Now, we show inductively that for every ψ, there exist constants c and d such that if $(C, O) \in [\![\psi]\!](S, i, j, \nu)$, then $|O| \leq c|C| + d$, i.e. the size of O is linear in the size of C.

- If $\psi = R$ `AS` x, then $c = d = 0$.
- $\psi = \rho$ `FILTER` $P(x_1, \ldots, x_n)$, and c', d' are the constants for ρ, then $c = c'$ and $d = d' + 1$.
- $\psi = \rho_1$ `OR` ρ_2, c_1, d_1 are the constants for ρ_1 and c_2, d_2 are the constants for ρ_2, then $c = \max(c_1, c_2)$ and $d = \max(d_1, d_2)$.
- $\psi = \rho_1 \,;\, \rho_2$, c_1, d_1 are the constants for ρ_1 and c_2, d_2 are the constants for ρ_2, then $c = \max(c_1, c_2)$ and $d = d_1 + d_2$.
- $\psi = \rho+$ and c', d' are the constants for ρ, then $c = c' + d'$ and $d = 0$.

The proof that the bounds above are correct goes by doing induction over the structure of the formula, and is straightforward in most of the cases. The case that needs more care is the $+$, which we now prove explicitly. Consider that $\psi = \rho+$, let c', d' be the constants for ρ, and consider some $(C, O) \in [\![\psi]\!](S, i, j, \nu)$. Then, by definition there exist l and a valuation ν' such that $(C, O) \in [\![\rho[l]]\!](S, i, j, \nu[\nu'/U])$. We prove the bound by doing a second induction over the number of iterations l of the definition of $+$. The base case is when $l = 1$ and the non-recursive definition is used, i.e., $(C, O) \in [\![\rho]\!](S, i, j, \nu[\nu'/U])$. In this case, the bound holds because $|O| \leq c'|C| + d' \leq (c' + d')|C| = c|C|$ (the first inequality holds from the first induction hypothesis and the second holds because $|C| \geq 1$).

For the inductive step, consider $l > 1$, meaning that $(C, O) \in [\![\rho \,;\, \rho[l-1]]\!](S, i, j, \nu[\nu'/U])$. Then, by definition there are k, C_1, C_2, O_1, O_2 such that $C = C_1 \cdot C_2$, $O = O_1 \cup O_2$, $(C_1, O_1) \in [\![\rho]\!](S, i, k, \nu)$ and $(C_2, O_2) \in [\![\rho[l-1]]\!](S, k+1, j, \nu)$. From the first induction hypothesis we get $|O_1| \leq c'|C_1| + d' \leq (c' + d')|C_1| = c|C_1|$; from the second induction we get $|O_2| \leq c|C_2| + d$. Joining both we get $|O| \leq |O_1| + |O_2| \leq c|C_1| + c|C_2| + d = c|C| + d$, thus the bound holds.

In particular, the bound of $|O|$ means that for every formula ψ in E-CEL($\mathcal{P}_=$), there is at most a linear number of comparisons between events, which is what we will exploit next.

By contradiction, assume that there is a formula ψ in E-CEL($\mathcal{P}_=$) that is equivalent to φ, and let c, d be the constants that bound the size of the comparing set. Then, for an arbitrary n consider the stream $S_n = R(1)R(2)\ldots R(n)T(n+1)T(n+2)\ldots T(2n)$, and consider the complex event $C_n = \{1, 2, \ldots, 2n\}$. It is clear that $C_n \in [\![\psi]\!](S_n)$. Therefore, there exist a comparing set O_n such that $(C_n, O_n) \in [\![\psi]\!](S_n)$. Now, define $O_n^{=}\{(i,j) \mid (P_=, i, j) \in O_n\}$ and $O_n^{\neq}\{(i,j) \mid (P_{\neq}, i, j) \in O_n\}$, i.e. the sets of pairs compared with equality and inequality, respectively. Because $|O_n|$ is linear in $|C_n|$, we know that, if n is sufficiently large, there exist positions k_1 and k_2 with $1 \leq k_1 \leq n < k_2 \leq 2n$ such that $(k_1, k_2) \notin O_n^{/} =$. Moreover, because all events have different value, we know that $|O_n^{=}| = 0$ (counting out the pairs of the form (k,k)). Now, define a new stream S'_n the same as S_n but replacing the values of $S_n[k_1]$ and $S_n[k_2]$ with some new value, e.g. $2n+1$. Then, all the comparisons made while evaluating (C_n, O_n) will still hold for S'_n, which means that $(C_n, O_n) \in [\![\psi]\!](S'_n)$ by following the same evaluation for S_n. But at the same time $k_1, k_2 \in C_n$, $\text{type}(S'_n[k_1]) = R$, $\text{type}(S'_n[k_2]) = T$ and $S'_n[k_1] = S'_n[k_2]$, which means that $C_n \notin [\![\psi]\!](S'_n)$, reaching a contradiction.

We conclude that there cannot exist a formula ψ in E-CEL($\mathcal{P}_=$) equivalent to φ.

A.3 Proof of Theorem 6

To prove the theorem we provide a construction that, for any formula $\varphi \in$ E-CEL($\mathcal{B}$), gives a formula $\psi \in$ S-CEL($\mathcal{B}^{\mathrm{S}}$) that is equivalent to φ. For notation, we will write $\rho' \subseteq \rho$ to say that ρ' is a subformula of ρ. Moreover, for a formula $\psi = \psi'$ `FILTER` $P(\bar{x}) \subseteq \varphi$ and $x \in \bar{x}$, we denote φ_x^P to be the subformula such that $\psi' \subseteq \varphi_x^P \subseteq \varphi$, $x \in \text{vdef}(\varphi_x^P)$ and there is no other $\rho \subset \varphi_x^P$ that satisfies the above. That is, φ_x^P is the formula closest to the filter such that x is defined in it.

Now, to simplify the proof we make some assumptions on φ. We assume that φ is safe, as defined in [18], that is, for every subformula of the form $\varphi_1 \,;\, \varphi_2$ it holds that $\text{vdef}_+(\varphi_1) \cap \text{vdef}_+(\varphi_2) = \emptyset$. We can make this assumption because in [18] they show that every E-CEL formula can be rewritten into a safe one. Without loss of generality, we assume that φ does not have unary predicates, as any predicate $P(x)$ can be easily simulated by a binary one with the form $P(x,x)$. Now, the construction is the following.

First, consider any subformula of φ of the form φ' `FILTER` $P(x,y)$ such that neither x nor y are in $\text{vdef}_+(\varphi')$. We will move $P(x,y)$ up to a position at which at least one of its variables is defined. For this, we use the notion of "well-formedness" defined in [18]. There, it is said that a formula is *well-formed* if for every subformula of the form ρ `FILTER` $P_1(x_1, \ldots, x_k)$ and every x_i, there is another subformula ρ_{x_i} such that $x_i \in \text{vdef}_+(\rho_{x_i})$. In fact, they use a more strict notion called "bound", which says that x must be bounded to ρ_x in the sense that it must be always defined (e.g. cannot appear only at one side of an `OR`). However, we will not make use of that property. We use the fact that φ is well-formed, which means that there are formulas φ_x and φ_y such that $x \in \text{vdef}_+(\varphi_x)$, $y \in \text{vdef}_+(\varphi_y)$ and $\varphi' \subseteq \varphi_x \subseteq \varphi_y \subseteq \varphi$ (wlog, we assume that $\varphi_x \subseteq \varphi_y$). Then, we can move up $P(x,y)$ by rewriting φ_x as $\varphi_x^\top$ `FILTER` $P(x,y)$ `OR` $\varphi_x^\perp$, where $\varphi_x^\top$ and $\varphi_x^\perp$ are φ_x replacing $P(x,y)$ with `TRUE` and `FALSE`, respectively. The idea is that $\varphi_x^\top$ `FILTER` $P(x,y)$ considers the cases where the condition $P(x,y)$ is needed and $\varphi_x^\perp$ adds the cases where it is not. As an intuition of why this is true, one can easily see that $[\![\varphi_x^\top \texttt{ FILTER } P(x,y)]\!](S,i,j,\nu) \cup [\![\varphi_x^\perp]\!](S,i,j,\nu) = [\![\varphi_x]\!](S,i,j,\nu)$. Now, let φ_1 be the result of doing this to every predicate of φ. Then, φ_1 is such that for every φ' `FILTER` $P(x,y) \subseteq \varphi_1$ it holds that either $x \in \text{vdef}_+(\varphi')$ or $y \in \text{vdef}_+(\varphi')$.

The intuition for the second step is that, for every filter $P(x,y)$, if at any moment of the evaluation we assigned x and y to two events, then they must satisfy $P(x,y)$. In order to achieve this, we want to rename each assignment with a new variable. The problem is that, if we do this right away, then it is not clear if we can replace the variables in the filters. For example, if we have the formula $(R\ \texttt{AS}\ x\ \texttt{OR}\ T\ \texttt{AS}\ x)\ \texttt{FILTER}\ P(x)$ and we rename both assignments with variables x_1 and x_2, then it is not clear which variable we need to use in the predicate P. To avoid this issue, we first rewrite φ_1 into a form called "disjunctive-normal form", defined in [18]. A formula φ is in *disjunctive-normal form* (or DNF) if $\varphi = (\varphi_1\ \texttt{OR}\ \ldots\ \texttt{OR}\ \varphi_n)$, where for each $i \in \{1, \ldots, n\}$, it is the case that:

- Every OR in φ_i occurs in the scope of a $+$-operator.
- For every subformula of φ_i of the form $(\varphi_i')+$, it is the case that φ_i' is in DNF.

In [18] it is shown that every formula φ in E-CEL can be translated into DNF. Moreover, one can see that the construction of the equivalent DNF formula shown there does not alter the property that we obtained from the previous step. Consider φ_1' to be the formula in DNF equivalent to φ_1. Now that the formula is safe and in DNF, we ensure that for every filter $P(x, y)$ in φ_1' the paths in the parse tree from the filter to the assignment of x and y never get inside an OR. This does not mean, however, that the assignments are not inside an OR in the whole formula. For example, $\rho = (R\ \texttt{AS}\ x\ \texttt{FILTER}\ P(x, y)\ ;\ S\ \texttt{AS}\ y)\ \texttt{OR}\ T\ \texttt{AS}\ y$ is a possible formula at this point, even though y appears in two sides of an OR. Instead, what we ensure is that for every filter $P(x_1, x_2)$, there is exactly one reachable assignment $R_i\ \texttt{AS}\ x_i$ for each variable x_i. In the case of ρ, the filter $P(x, y)$ can reach only the x in $T\ \texttt{AS}\ x$ and the y in $S\ \texttt{AS}\ y$, but not the one in $T\ \texttt{AS}\ y$. Now that for every filter there is exactly one assignment for each of its variables, we can then rename the variables safely. For this, identify each assignment $R\ \texttt{AS}\ x$ of φ_1' with a unique id i, and for every filter $P(x, y)$ let i_x^P and i_y^P be the ids of the assignments reachable from that filter. Then, we rewrite φ_1' using a new set of variables $\{x_1, x_2, \ldots\}$ in the following way:

- Replace each assignment $R\ \texttt{AS}\ y$ with $R\ \texttt{AS}\ x_i$, where i is the id of the assignment,
- Replace each filter $P(x, y)$ with $P(x_{i_x^P}, x_{i_y^P})$.

Call φ_2 the resulting formula. Because φ_1' was safe and in DNF, then the renaming does not change the semantics.

The final step is to turn φ_2 into a S-CEL formula. After turning φ into φ_2, we claim that now we can do it safely by pushing each predicate up until it reaches a point where all its variables are assigned. Formally, considering labels $\{A_1, A_2, \ldots\}$, what we do is:

- Replace each assignment $R\ \texttt{AS}\ x_i$ with $R\ \texttt{IN}\ A_i$,
- For each subformula with a filter $P(x_i, x_j)$, remove de filter and instead add the filter $P^{\mathrm{S}}(A_i, A_j)$ at formula $\varphi_{x_j}^P$ (assuming $\varphi_{x_i}^P \subseteq \varphi_{x_j}^P$).

Then, we define ψ as the resulting formula. The intuition of why ψ keeps the same semantics of φ_2 is the following. At φ_2 we know that for every $\varphi'\ \texttt{FILTER}\ P(x_i, x_j) \subseteq \varphi_2$, one variable (e.g. x_i) is assigned in, and only in φ', while the other is assigned somewhere else in the formula, and only there. Then, for every evaluation that at some point needs to assign x_i to some events e_i, it must get inside of φ' (because x_i is only there), thus it must first satisfy the filter, i.e. $P(e_i, e_j)$ must hold, where e_j is the current assignment of x_j. Moreover, x_j is only named once in $\varphi_{x_j}^P$, and since x_j is defined in $\varphi_{x_j}^P$ (it cannot be inside a $+$), it holds that every assignment of x_i (which is only one) and every assignment of x_j must satisfy $P(x_i, x_j)$. Since all the assignments of x_i and x_j were labelled with A_i and A_j, respectively, this is exactly what $P^{\mathrm{S}}(A_i, A_j)$ in ψ represents. Thus, we claim that the resulting formula ψ is equivalent to φ.

A.4 Proof of Theorem 7

To prove the theorem, we show that the formula:

$$\varphi \;=\; R \texttt{ AS } x \,;\, ((S \texttt{ AS } y \,;\, T \texttt{ AS } z) \texttt{ FILTER } (x = y + z))+$$

in E-CEL($\mathcal{P}_+$) is not expressible in S-CEL($\mathcal{P}_+^{\mathrm{S}}$). We begin by giving some definitions we will need next. We define an *evaluation tree* T as an ordered unranked tree where each node t has a valuation μ_t associated to it. Moreover, let $C_t = \sup(\mu_t)$, that is, C_t contains all the positions that appear in the valuation μ_t. For each node t, we refer to the children $t_1, t_2, \ldots, t_k$ of t as $\text{children}(t) = (t_1, t_2, \ldots t_k)$. Notice that we do not bound the number of children of a node, and that there is an order between the children. We often refer to T by its root node, e.g. if t is the root of T, then $\mu_T = \mu_t$ and $\text{children}(T) = \text{children}(t)$. With the notion of evaluation tree, we extend the semantics of S-CEL in the following way. We say that T belongs to the evaluation trees of ψ over S starting at position i and ending at j (denoted by $T \in [\![\psi]\!]^{\text{tree}}(S, i, j)$) if one of the following conditions holds:

- $\psi = R$, $\text{children}(T) = ()$, $\mu_T(R) = \{j\}$, $\text{type}(S[j]) = R$ and $\mu(A) = \emptyset$ for every $A \neq R$.
- $\psi = \rho \texttt{ IN } A$, $\text{children}(T) = (t)$, $t \in [\![\rho]\!]^{\text{tree}}(S, i, j)$, $\mu_T(A) = C_t$ and $\mu_T(B) = \mu_t(B)$ for all $B \neq A$.
- $\psi = \rho \texttt{ FILTER } P(X_1, \ldots, X_n)$, $T \in [\![\rho]\!]^{\text{tree}}(S, i, j)$ and $(S[\mu_T(X_1)], \ldots, S[\mu_T(X_n)]) \in P$.
- $\psi = \rho_1 \texttt{ OR } \rho_2$ and either $T \in [\![\rho_1]\!]^{\text{tree}}(S, i, j)$ or $T \in [\![\rho_2]\!]^{\text{tree}}(S, i, j)$
- $\psi = \rho_1 \,;\, \rho_2$, $\text{children}(T) = (t_1, t_2)$ and there exists $k \in \mathbb{N}$ such that $\mu_T = \mu_{t_1} \cup \mu_{t_2}$, $t_1 \in [\![\rho_1]\!]^{\text{tree}}(S, i, k)$ and $t_2 \in [\![\rho_2]\!]^{\text{tree}}(S, k+1, j)$.
- $\psi = \rho+$, $\text{children}(T) = (t_1, t_2, \ldots, t_l)$ and there exist $k_1, \ldots, k_l$ with $k_l = j$ such that $\mu_T = \mu_{t_1} \cup \ldots \cup \mu_{t_l}$, $t_1 \in [\![\rho]\!]^{\text{tree}}(S, i, k_1)$ and $t_i \in [\![\rho]\!]^{\text{tree}}(S, k_{i-1}+1, k_i)$.

The behaviour of the tree semantics is the same as in the original semantics, with the difference that, instead of returning a complex event C, it returns the tree with the evaluations that were used in the construction of C. Thus, it is easy to see that:

- If $T \in [\![\psi]\!]^{\text{tree}}(S, i, j)$ then $C_T \in [\![\psi]\!](S, i, j, \mu_T)$, and
- If $C \in [\![\psi]\!](S, i, j, \mu)$ then there exists some $T \in [\![\psi]\!]^{\text{tree}}(S, i, j)$ with $\mu_T = \mu$.

We use the evaluation tree because it gives us more information about how the complex event was evaluated than the complex event itself.

The following lemmas exhibit some interesting properties about the evaluation trees, which can be easily proven:

▶ **Lemma 17.** *For every formula ψ in S-CEL there exists some N such that every $T \in [\![\psi]\!]^{\text{tree}}(S, i, j)$ is of depth at most N, for any stream S and positions i, j.*

▶ **Lemma 18.** *Consider a formula ψ in S-CEL, a stream S and positions i, j. For every $T \in [\![\psi]\!]^{\text{tree}}(S, i, j)$ and $k \in C_T$ there is exactly one leaf t in T such that $k \in C_t$. Moreover, the only nodes t' in T with $k \in C_{t'}$ are the ones in the path between t and T.*

Now we are ready to prove that φ does not have an equivalent formula in S-CEL($\mathcal{P}_+^{\mathrm{S}}$). By contradiction, assume that there exists such formula, call it ψ. Let D be the maximum depth of the evaluation trees of ψ. Now, for an arbitrary N consider the stream:

$$S \;=\; \begin{array}{cccccccc} R & S & T & S & T & \cdots & S & T & \cdots \\ 2N & 1 & 2N-1 & 3 & 2N-3 & & N-1 & N+1 & \end{array}$$

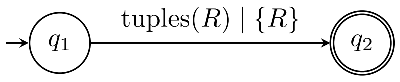

Figure 3 A CEA for R with the $*$-semantics.

Intuitively, we chose this stream because the only triples that satisfy $X = Y + Z$ are the ones where the set X is associated to the only R event and the sets Y and Z are associated, one to only one event S, and the other to only the event T after that S. Clearly the complex event $C = \{1, 2, \ldots, N, N+1\}$ is in $[\![\varphi]\!](S, 1, N+1, \nu)$ for some ν. As ψ must be equivalent to φ, there must be some tree $T \in [\![\psi]\!]^{\text{tree}}(S, 1, N+1)$ such that $C_T = C$. Let t be the leaf of T that contains the position 1 (i.e. the only R-tuple), and let $t_1, t_2, \ldots, t_d$ be the nodes in the path from t to T ($d \leq D$). We know that the sum predicate was applied at most d times in the path between t and T. Moreover, for every occurrence of the sum predicate $A_i = B_i + C_i$ at some node $t_1, \ldots, t_d$, it can only be satisfied if $|\mu_{t_i}(A_i)| = |\mu_{t_i}(B_i)| = |\mu_{t_i}(C_i)| = 1$, and, in particular, if $\mu_{t_i}(A_i) = \{1\}$, $\mu_{t_i}(B_i) = \{r_1\}$ and $\mu_{t_i}(C_i) = \{r_2\}$ for some r_1, r_2 at distance 1. We do not consider the case where some of the labels are mapped to $\emptyset$ because in that case the predicate is not filtering anything. As stated before, it is easy to see that any other scenario does not satisfy the predicate.

Define O as the set that contains all positions that were compared with $S[1]$, i.e. $O = \{l \mid \exists i.\ l \in B_i \cup C_i\}$. Since there were at most d occurrences of the sum predicate and $|B_i| = |C_i| = 1$ for all i, we know that $|O|$ is at most $2d$. Then, if we choose an N big enough (e.g. $N = 2d + 1$), we can find a position i that is in C but not in O. Intuitively, this means that $S[i]$ was not compared with $S[1]$. Moreover, note that all the events except the first one have odd values, so we know that $S[i]$ was not compared with any other event. Wlog, assume that $S[i]$ is of type S. Then, we can define a new stream S' the same as S but replacing the value of $S[i]$ with any other value, say 0, and the evaluation tree T would still be in $[\![\psi]\!]^{\text{tree}}(S', 1, N+1)$, thus $C_T \in [\![\psi]\!](S', 1, N+1, \mu_T)$ and $C_T = C \in [\![\varphi]\!](S', 1, N+1, \nu)$ for some ν. But since the sum of the values $S[i]$ (of type S) and $S[i+1]$ (of type T) is not N, we know that C cannot be in $[\![\varphi]\!](S', 1, N+1, \nu)$, reaching a contradiction.

B Proofs of Section 5

B.1 Proof of Theorem 8

Let $\mathcal{U}$ be a set of unary predicates and let φ be a formula in S-CEL($\mathcal{U}^{\text{S}}$). We start by showing how to construct a CEA $\mathcal{A}_\varphi$ over $\mathcal{U}$ that is equivalent to φ. For the construction, we extend the transitions of CEA with labels instead of $\bullet$ and $\circ$ marks, that is, now a transition will have the form (p, P, L, q), where p, q are states, $P \in \mathcal{U}$ and $L \subseteq \mathbf{L}$. Intuitively, the set of labels represents the labels assigned to the event read when taking that transition. We use this as an auxiliary model, and at the end of the construction we return to the original CEA transitions. We proceed by induction, assuming that for every formula ψ shorter than φ there is a CEA $\mathcal{A}_\psi$ that is equivalent to ψ:

- If $\varphi = R$, then $\mathcal{A}_\varphi$ is defined as depicted in figure 3, i.e. $\mathcal{A}_\varphi = (\{q_1, q_2\}, \Delta_\varphi, \{q_1\}, \{q_2\})$ with $\Delta_\varphi = \{(q_1, \text{tuples}(R), \{R\}, q_2)\}$
- If $\varphi = \psi$ `IN` A, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where Δ_φ is the result of adding label A to all non-empty transitions of Δ_ψ. Formally, $\Delta_\varphi = \{(p, P, L, q) \in \Delta_\psi \mid L = \emptyset\} \cup \{(p, P, L, q) \mid$

$\exists L' \neq \emptyset$ such that $(p, P, L', q) \in \Delta_\psi \wedge L = L' \cup \{A\}\}$.

- If $\varphi = \psi$ `FILTER` $P^{\mathrm{S}}(A)$ for some unary event predicate P and $A \in \mathbf{L}$, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where Δ_φ is defined as $\{(p, P', L, q) \in \Delta_\psi \mid A \notin L\} \cup \{(p, P \wedge P', L, q) \mid (p, P', L, q) \in \Delta_\psi \wedge A \in L\}$. The intuition behind this is that since P^{S} is the universal extension of P, all tuples that are labeled by A must satisfy P.
- If $\varphi = \psi_1$ `OR` ψ_2, then $\mathcal{A}_\varphi$ is the automata union between $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$ as one would expect: $\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_{\psi_1} \cup \Delta_{\psi_2}, I_{\psi_1} \cup I_{\psi_2}, F_{\psi_1} \cup F_{\psi_2})$.
- If $\varphi = \psi_1 \,;\, \psi_2$, then $\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_\varphi, I_{\psi_1}, F_{\psi_2})$ where $\Delta_\varphi = \Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(p, P, L, q) \mid q \in I_{\psi_2} \wedge \exists q' \in F_{\psi_1}.(p, P, L, q') \in \Delta_{\psi_1}\}$.
- If $\varphi = \psi+$, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where $\Delta_\varphi = \Delta_\psi \cup \{(p, P, L, q) \mid q \in I_\psi \wedge \exists q' \in F_\psi.(p, P, L, q') \in \Delta_\psi\}$.

Finally, to return to the original marking transitions, each transition $t = (p, P, L, q)$ will be turned into a transition (p, P, m, q) such that $m = \bullet$ iff $L \neq \emptyset$ and $m = \circ$ if $L = \emptyset$. Notice that the size of the resulting automaton $\mathcal{A}_\varphi$ is linear in the size of φ. Moreover, unlike the construction in [18], where it needed some preprocessing on the formula φ (in particular, to push the predicates down), here we do not need any preprocessing because the construction for the case `FILTER` is straightforward.

Now, we prove the second direction, that is, for every CEA $(\mathcal{U})$ $\mathcal{A}$, there exists a formula φ in S-CEL($\mathcal{U}^{\mathrm{S}}$) such that $[\![\mathcal{A}]\!]^*_n(S) = [\![\varphi]\!]_n(S)$ for every stream S and $n \in \mathbb{N}$. For this, we give a construction that, for any $\mathcal{A} = (Q, \Delta, I, F)$, defines a formula $\varphi_{\mathcal{A}}$.

Consider the set $Q = \{q_1, q_2, \ldots, q_n\}$. To simplify the construction, assume that $I = \{q_1\}$ and $F = \{q_n\}$, and define the `FALSE` formula as a formula that is never satisfied. One way to define it is `FALSE` $= (R$ `FILTER` $\emptyset)$, but we will use `FALSE` to keep the proof simple. Moreover, we only consider non-zero executions, that is, an accepting run of a CEA must be of length at least 1. Notice that these limits automata to not being able to run over the empty stream, but since we only care about large (potentially infinite) streams, this case is not of interest.

The main idea is based on the construction from finite automata over words which defines, for every pair of states q_i, q_j, a S-CEL formula φ_{ij} that represents the complex events defined by the $*$-runs from q_i to q_j. Furthermore, we define φ^k_{ij} the same way but with the restriction that the $*$-runs only pass through states $q_1, \ldots, q_k$. Given that, it is clear then that $\varphi^{|Q|}_{ij} = \varphi_{ij}$.

We define φ^k_{ij} recursively in the following way. The base case $k = 0$ is defined as follows. For each i, j, if there is no transition from q_i to q_j in $\mathcal{A}$, then $\varphi^0_{ij} =$ `FALSE`; otherwise we define it as:

$$\varphi^0_{ij} = \rho_{P_1} \texttt{ OR } \rho_{P_2} \texttt{ OR } \ldots \texttt{ OR } \rho_{P_k}$$

where $P_1, \ldots, P_k$ are all the predicates of the $\bullet$-transitions from q_i to q_j. Moreover, ρ_P represents the S-CEL formula that accepts all complex events that consist of a single event that satisfies P, defined as $\rho_P := (R_1$ `FILTER` $P)$ `OR` $\cdots$ `OR` $(R_r$ `FILTER` $P)$ for $\mathcal{R} = \{R_1, \ldots, R_r\}$. Note that at each R_i `FILTER` P we need to remove the predicates of the form tuples(R), which is done as expected by replacing each one with either `TRUE` if $R = R_i$ or `FALSE` if $R \neq R_i$. Next, the recursion is defined as:

$$\varphi^k_{ij} \;=\; \varphi^{k-1}_{ij} \texttt{ OR } (\varphi^{k-1}_{ik} \,;\, \varphi^{k-1}_{kj}) \texttt{ OR } (\varphi^{k-1}_{ik} \,;\, \varphi^{k-1}_{kk}+ \,;\, \varphi^{k-1}_{kj})$$

Finally, the final formula $\varphi_{\mathcal{A}}$ is the result of considering φ_{1n}. The correctness of the construction can be proven by doing induction over the number of states.

B.2 Proof of Proposition 9

Consider any CEA($\mathcal{U}$) $\mathcal{A} = (Q, \Delta, I, F)$ that has the $*$-property. Now, define the CEA($\mathcal{U}$) $\mathcal{A}' = (Q, \Delta^\bullet, I, F)$ such that $\Delta^\bullet = \{(p, P, \bullet, q) \mid (p, P, \bullet, q) \in \Delta\}$. Let S be any stream. We now prove that $[\![\mathcal{A}]\!]_n(S) = [\![\mathcal{A}']\!]^*_n(S)$. First, consider a complex event $C \in [\![\mathcal{A}]\!]_n(S)$. This means that there is an accepting run of $\mathcal{A}$:

$$\rho : q_0 \xrightarrow{P_1/m_1} q_1 \xrightarrow{P_2/m_2} \cdots \xrightarrow{P_n/m_n} q_n$$

Such that $C_\rho = C$. Let $C = \{i_1, i_2, \ldots, i_k\}$, and consider the stream $S[C\rangle$ as the stream formed by the events $S[i_1]S[i_2]\ldots S[i_k]$. Then, because $\mathcal{A}$ defines a function with $*$-property, there has to be an accepting run of $\mathcal{A}$ over $S[C\rangle$.

$$\rho' : q'_0 \xrightarrow{P'_1/\bullet} q'_1 \xrightarrow{P'_2/\bullet} \cdots \xrightarrow{P'_k/\bullet} q'_k$$

Because of the construction, the analogous $*$-run of $\mathcal{A}'$ over $S[C\rangle$:

$$\sigma' : (q'_0, 0) \xrightarrow{P'_1/\bullet} (q'_1, 1) \xrightarrow{P'_2/\bullet} \cdots \xrightarrow{P'_k/\bullet} (q'_k, k)$$

is an accepting $*$-run. Moreover, one can unfold $S[C\rangle$ back to the original stream S and the $*$-run:

$$\sigma : (q'_0, 0) \xrightarrow{P'_1/\bullet} (q'_1, i_1) \xrightarrow{P'_2/\bullet} \cdots \xrightarrow{P'_k/\bullet} (q'_k, i_k)$$

is an accepting $*$-run of $\mathcal{A}'$ over S, therefore $C_\sigma = C \in [\![\mathcal{A}']\!]^*_n(S)$.

The proof for the converse case is practically the same. Assume that $C \in [\![\mathcal{A}']\!]^*_n(S)$, which means that the $*$-run σ of $\mathcal{A}'$ over S exists. Moreover, the $*$-run σ' of $\mathcal{A}'$ over $S[C\rangle$ also exists, thus by the construction of $\mathcal{A}'$, the run ρ' of $\mathcal{A}$ must exist. Because $\mathcal{A}$ defines a function with $*$-property, the accepting run ρ of $\mathcal{A}$ over S has to exist. We conclude that $C_\rho = C \in [\![\mathcal{A}]\!]_n(S)$.

C Proofs of Section 6

C.1 Proof of Theorem 13

C.1.1 Construction of CEA

Let $\mathcal{U}$ be a set of unary predicates and let φ be a formula in S-CEL+($\mathcal{U}^{\mathrm{S}}$). We start by showing how to construct a CEA $\mathcal{A}_\varphi$ over $\mathcal{U}$ that is equivalent to φ. To prove this, we use a construction similar to the one in Theorem 8. As in Theorem 8, we extend the transitions of CEA with labels instead of $\bullet$ and $\circ$ marks, and then return to the original CEA transitions. We proceed by induction, assuming that for every formula ψ shorter than φ there is a CEA $\mathcal{A}_\psi$ that is equivalent to ψ:

- If $\varphi = R$, then $\mathcal{A}_\varphi$ is defined as depicted in figure 4, i.e. $\mathcal{A}_\varphi = (\{q_1, q_2\}, \Delta_\varphi, \{q_1\}, \{q_2\})$ with $\Delta_\varphi = \{(q_1, \texttt{TRUE}, \emptyset, q_1), (q_1, \text{tuples}(R), \{R\}, q_2)\}$
- If $\varphi = \psi$ `IN` A, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where Δ_φ is the result of adding label A to all non-empty transitions of Δ_ψ. Formally, $\Delta_\varphi = \{(p, P, L, q) \in \Delta_\psi \mid L = \emptyset\} \cup \{(p, P, L, q) \mid \exists L' \neq \emptyset \text{ such that } (p, P, L', q) \in \Delta_\psi \wedge L = L' \cup \{A\}\}$.
- If $\varphi = \pi_L(\psi)$ for some $L \subseteq \mathbf{L}$, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where Δ_φ is the result of projecting the labels of each transition $t \in \Delta_\psi$ with L. Formally, that is $\Delta_\varphi = \{(p, P, L \cap L', q) \mid (p, P, L', q) \in \Delta_\psi\}$.

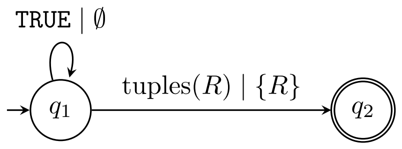

Figure 4 A CEA for the atomic formula R.

- If $\varphi = \psi$ FILTER $P^{\mathrm{S}}(A)$ for some unary event predicate P and $A \in \mathbf{L}$, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where Δ_φ is defined as $\{(p, P', L, q) \in \Delta_\psi \mid A \notin L\} \cup \{(p, P \wedge P', L, q) \mid (p, P', L, q) \in \Delta_\psi \wedge A \in L\}$. The intuition behind this is that since P^{S} is the universal extension of P, all tuples that are labeled by A must satisfy P.
- If $\varphi = \psi_1 \,;\, \psi_2$, then $\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_\varphi, I_{\psi_1}, F_{\psi_2})$ where $\Delta_\varphi = \Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(p, P, L, q) \mid q \in I_{\psi_2} \wedge \exists q' \in F_{\psi_1}.(p, P, L, q') \in \Delta_{\psi_1}\}$.
- If $\varphi = \psi_1 : \psi_2$, then we do the following. We add a new dummy state q, which will make the connection between the first part and the second. In order to obtain the $:$ semantics, we will restrict q to only arrive and depart non-empty transitions. We define $\mathcal{A}_\varphi = (Q_\varphi, \Delta_\varphi, I_\varphi, F_\varphi)$ as follows. First, the set of states is $Q_\varphi = Q_{\psi_1} \cup Q_{\psi_2} \cup \{q\}$, where q is a new dummy state. Then, the transition relation is $\Delta_\varphi = \Delta_{\psi_1} \cup \Delta_{\psi_2} \cup \{(q_1, P, L, q) \mid L \neq \emptyset \wedge \exists q' \in F_{\psi_1}.\ ((q_1, P, L, q') \in \Delta_{\psi_1})\} \cup \{(q, P, L, q_2) \mid L \neq \emptyset \wedge \exists q' \in I_{\psi_2}.\ ((q', P, L, q_2) \in \Delta_{\psi_1})\}$. Finally, the sets of initial and final states are $I_\varphi = I_{\psi_1}$ and $F_\varphi = F_{\psi_2}$. The idea of why this works is that at some point it has to go from $\mathcal{A}_{\psi_1}$ to $\mathcal{A}_{\psi_2}$, and the only way to do it is through q. Then, because q only receives and departs non-empty transitions, we get the desired result.
- If $\varphi = \psi+$, then $\mathcal{A}_\varphi = (Q_\psi, \Delta_\varphi, I_\psi, F_\psi)$ where $\Delta_\varphi = \Delta_\psi \cup \{(p, P, L, q) \mid q \in I_\psi \wedge \exists q' \in F_\psi.(p, P, L, q') \in \Delta_\psi\}$.
- If $\varphi = \psi\oplus$, then we can use an idea similar to the one we used for the operator :. We add a new dummy state q, which will make the connection between one iteration and the next one. In order to obtain the $\oplus$ semantics, we will restrict q to only arrive and depart non-empty transitions. We do this as follows: we define $\mathcal{A}_\varphi = (Q_\psi \cup \{q\}, \Delta_\varphi, I_\psi, F_\psi)$. The transition relation is $\Delta_\varphi = \Delta_\psi \cup \{(q_1, P, L, q) \mid L \neq \emptyset \wedge \exists q' \in F_\psi.\ ((q_1, P, L, q') \in \Delta_\psi)\} \cup \{(q, P, L, q_2) \mid L \neq \emptyset \wedge \exists q' \in I_\psi.\ ((q', P, L, q_2) \in \Delta_\psi)\}$. The idea of why this works is the same one for the : case: at some point it has to go from one iteration to another, and the only way to do it is through q. Then, because q only receives and departs non-empty transitions, we get the desired result.
- If $\varphi =$ START(ψ), then we need to force the first transition to contain at least one label. Similar to the previous cases, we add a new dummy state q which will work as our initial state, and we will restrict it to only depart non-empty transitions. Formally, $\mathcal{A}_\varphi = (Q_\psi \cup \{q\}, \Delta_\varphi, \{q\}, F_\psi)$, where $\Delta_\varphi = \Delta_\psi \cup \{(q, P, L, p) \mid L \neq \emptyset \wedge \exists q' \in I_\psi.((q', P, L, p) \in \Delta_\psi)\}$.
- If $\varphi = \psi_1$ OR ψ_2, then $\mathcal{A}_\varphi$ is the automata union between $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$ as one would expect: $\mathcal{A}_\varphi = (Q_{\psi_1} \cup Q_{\psi_2}, \Delta_{\psi_1} \cup \Delta_{\psi_2}, I_{\psi_1} \cup I_{\psi_2}, F_{\psi_1} \cup F_{\psi_2})$.

Finally, to return to the original marking transitions, each transition $t = (p, P, L, q)$ will be turned into a transition (p, P, m, q) such that $m = \bullet$ iff $L \neq \emptyset$ and $m = \circ$ if $L = \emptyset$. Notice that the size of the resulting automaton $\mathcal{A}_\varphi$ is linear in the size of φ.

C.1.2 From CEA to unary S-CEL

Now we proceed to show the opposite direction: given a CEA, define an equivalent unary S-CEL formula. Let $\mathcal{A} = (Q, \Delta, I, F)$ be a CEA, with $Q = \{q_1, \ldots, q_n\}$. Without loss of generality, assume that there is only one initial state and one final state, i.e., $I = \{q_1\}$ and $F = \{q_n\}$. Here we use the same idea of Theorem 8: to define, for every pair of states (q_i, q_j), a formula φ_{ij} that represents the complex events of the runs from q_i to q_j. Furthermore, we define φ^k_{ij} the same way but with the restriction that the runs only pass through states $q_1, \ldots, q_k$.

We define φ^k_{ij} recursively in the following way. The base case $k = 0$ is defined as follows. For each i, j, if there is no transition from q_i to q_j in $\mathcal{A}$, then $\varphi^0_{ij} = \texttt{FALSE}$ (as defined in Theorem 8); otherwise we define it as:

$$\varphi^0_{ij} = (\rho_{P_1}\ \texttt{OR}\ \ldots\ \texttt{OR}\ \rho_{P_m})\ \texttt{IN}\ L_\bullet\ \texttt{OR}\ (\rho_{U_1}\ \texttt{OR}\ \ldots\ \texttt{OR}\ \rho_{U_n})$$

where $\{P_k\}$ are all the predicates of $\bullet$-transitions $(q_i, P_k, \bullet, q_j)$ and $\{U_k\}$ are all the predicates of $\circ$-transitions $(q_i, U_k, \circ, q_j)$. Moreover, ρ_P represents the S-CEL formula that accepts all complex events that consist of a single event that satisfies P, defined as $\rho_P = (R_1\ \texttt{FILTER}\ P)\ \texttt{OR}\ \cdots\ \texttt{OR}\ (R_r\ \texttt{FILTER}\ P)$ for $\mathcal{R} = \{R_1, \ldots, R_r\}$ (like in Theorem 8).

Next, the recursion is defined as:

$$\varphi^k_{ij}\ =\ \varphi^{k-1}_{ij}\ \texttt{OR}\ (\varphi^{k-1}_{ik} : \varphi^{k-1}_{kj})\ \texttt{OR}\ (\varphi^{k-1}_{ik} : \varphi^{k-1}_{kk} +\ : \varphi^{k-1}_{kj})$$

Finally, the final formula $\varphi_{\mathcal{A}}$ is the result of considering φ_{1n}, forcing it to begin immediately at position 0, and then projecting to retrieve only the events with label $L_\bullet$ (the ones that come from $\bullet$ transitions). Formally, we define it as $\varphi_{\mathcal{A}} := \pi_{\{L_\bullet\}}(\texttt{START}(\varphi_{1n}))$.

Finally, it is straightforward to prove the correctness of the construction by induction over the number of states.

C.2 Proof of Proposition 14

Consider a formula φ in S-CEL $\cup\{\ :\ , \oplus\}(\mathcal{U})$. We first prove that there is a formula ψ in S-CEL $\cup\{\ :\ , \texttt{STRICT}\}(\mathcal{U})$ which is equivalent to φ, and then the proof follows directly from Theorem 16.

Consider any formula φ' in S-CEL $\cup\{\ :\ , \texttt{STRICT}\}(\mathcal{U})$. We prove by induction that there exists a formula ψ' in S-CEL $\cup\{\ :\ , \texttt{STRICT}\}(\mathcal{U})$ which is equivalent to $\varphi'\oplus$. For simplicity, we assume that all the filters are applied at the bottom-most level, which can be achieved by using the same idea as in Theorem 4. Moreover, we can then drop the `IN` labellings, since all the filters are at the assignment level and thus the labels do not alter the query semantics.

Now that all filters in our φ' are at the bottom-most level and all the `IN` are dropped, we proceed to prove by induction that there exists a formula ψ' in S-CEL $\cup\{\ :\ , \texttt{STRICT}\}(\mathcal{U})$ which is equivalent to $\varphi'\oplus$. We consider the possible cases for φ':

- For the base case, if $\varphi' = R$, then $\psi' = \texttt{STRICT}(R+)$ is equivalent to $\varphi'\oplus$. Similarly for the case $\varphi' = R\ \texttt{FILTER}\ P(R)$.
- If $\varphi' = \texttt{STRICT}(\varphi_1)$, then $\psi' = \texttt{STRICT}(\varphi_1+)$ is equivalent to $\varphi'\oplus$.
- If $\varphi' = \varphi_1 \mathbin{;} \varphi_2$, then $\psi' = \varphi_1 \mathbin{;} (\varphi_2\ \texttt{OR}\ ((\varphi_2 : \varphi_1)+ \mathbin{;} \varphi_2))$ is equivalent to $\varphi'\oplus$.
- If $\varphi' = \varphi_1+$, then $\psi' = \varphi_1+$ is equivalent to $\varphi'\oplus$.

We do not consider the $:$ -case since we know that they can be removed by using `STRICT` instead.

The last and more complex operator is the `OR`, for which we have to consider $\varphi' = \varphi_1 \texttt{ OR } \varphi_2$, with all possible cases for φ_1 and φ_2. The simplest scenario is where both φ_1 and φ_2 have either the form R, $R \texttt{ FILTER } P(R)$ or $\texttt{STRICT}(\psi)$ for some ψ, at which case we can simply write $\varphi'\oplus$ as $\texttt{STRICT}(\varphi'+)$.

Now we consider the cases where some of them does not have this form (w.l.o.g. assume is φ_2). Consider first the case $\varphi_2 = \rho_1 \,;\, \rho_2$. Here we use the following equivalence:

$$
\begin{aligned}
(\varphi_1 \texttt{ OR } (\rho_1 \,;\, \rho_2))\oplus \quad \equiv \quad & \varphi_1 \oplus \;\; \texttt{OR } (\rho_1 \,;\, \rho_2) \oplus \;\; \texttt{OR} & (1)\\
& (\varphi_1 \oplus \;\; : (\rho_1 \,;\, \rho_2)\oplus) \oplus \;\; \texttt{OR} & (2)\\
& \varphi_1 \oplus \;\; : ((\rho_1 \,;\, \rho_2) \oplus \;\; : \varphi_1\oplus) \oplus \;\; \texttt{OR} & (3)\\
& ((\rho_1 \,;\, \rho_2) \oplus \;\; : \varphi_1\oplus) \oplus \;\; \texttt{OR} & (4)\\
& (\rho_1 \,;\, \rho_2) \oplus \;\; : (\varphi_1 \oplus \;\; : (\rho_1 \,;\, \rho_2))\oplus & (5)
\end{aligned}
$$

Here, part (1) has no problem since the $\oplus$-operator is applied over subformulas of the original one, thus by induction hypothesis they can be written without $\oplus$. Moreover, with some basic transformations in part (2) (replacing $(\rho_1 \,;\, \rho_2)\oplus$ by $\rho_1 \,;\, (\rho_2 \texttt{ OR } ((\rho_2 : \rho_1)+ \,;\, \rho_2)))$ one can show that it is equivalent to the formula $(\sigma_1 \,;\, \sigma_2)\oplus$, where $\sigma_1 = \varphi_1 \oplus \;\; : \rho_1$ and $\sigma_2 = \rho_2 \texttt{ OR } ((\rho_2 : \rho_1)+ \,;\, \rho_2)$. Then, we can replace $(\sigma_1 \,;\, \sigma_2)$ with $\sigma_1 \,;\, (\sigma_2 \texttt{ OR } ((\sigma_2 : \sigma_1)+ \,;\, \sigma_2))$, and the resulting formula will contain only one $\oplus$ in the form $\varphi_1\oplus$, which by induction hypothesis can also be removed. Similarly, parts (3), (4) and (5) can be rewritten this way, therefore for the case of $\varphi_2 = \rho_1 \,;\, \rho_2$ the induction statement remains true.

Now consider the case $\varphi_2 = \rho+$. Notice that the following equivalence regarding $+$ holds: $\rho+ \equiv \rho \texttt{ OR } \rho \,;\, \rho+$. Thus, φ' can then be written as $(\varphi_1 \texttt{ OR } \rho) \texttt{ OR } (\rho \,;\, \rho+)$. Then, if we redefine $\varphi_1 := (\varphi_1 \texttt{ OR } \rho)$ and $\varphi_2 = (\rho \,;\, \rho+)$, clearly φ' would have the form $\varphi_1 \texttt{ OR } (\rho_1 \,;\, \rho_2)$. Therefore, we can apply the previous case and the resulting formula will still satisfy the induction statement, thus it remains true in the case $\varphi_2 = \rho+$.

Then, we can replace every subformula $\varphi'\oplus$ of φ with its equivalent formula ψ' in a bottom-up fashion to ensure that each φ' is in S-CEL$\,\cup\{\;:\;, \texttt{STRICT}\}(\mathcal{U})$. Finally, the remaining formula ψ does not contain $\oplus$ and thus is in S-CEL$\,\cup\{\;:\;, \texttt{STRICT}\}(\mathcal{U})$. The converse case follows directly from Proposition 15.

C.3 Proof of Proposition 15

Consider a formula φ in S-CEL$\,\cup\{\texttt{STRICT}\}(\mathcal{P})$. We first prove that for every formula φ' in S-CEL$\,\cup\{\texttt{STRICT}\}(\mathcal{P})$ there is a formula ψ' in S-CEL$\,\cup\{\;:\;, \oplus\}(\mathcal{P})$ that is equivalent to $\texttt{STRICT}(\varphi')$, for which we do induction over the structure of φ'. The base case is $\varphi' = R$, which already satisfies the above considering $\psi' = R$. For the inductive step, consider the following cases.

- If $\varphi' = \rho \texttt{ IN } A$, then $\texttt{STRICT}(\varphi') \equiv \texttt{STRICT}(\rho) \texttt{ IN } A$. By induction hypothesis, there is a formula σ in S-CEL$\,\cup\{\;:\;, \oplus\}(\mathcal{P})$ equivalent to $\texttt{STRICT}(\rho)$. Thus, $\psi' = \sigma \texttt{ IN } A$ is equivalent to $\texttt{STRICT}(\varphi')$.
- If $\varphi' = \rho_1 \,;\, \rho_2$, then $\texttt{STRICT}(\varphi') \equiv \texttt{STRICT}(\rho_1) : \texttt{STRICT}(\rho_2)$. By induction hypothesis, both $\texttt{STRICT}(\rho_1)$ and $\texttt{STRICT}(\rho_2)$ have equivalent formulas σ_1 and σ_2, respectively, in S-CEL$\,\cup\{\;:\;, \oplus\}(\mathcal{P})$. Thus, $\psi' = \sigma_1 : \sigma_2$ is equivalent to $\texttt{STRICT}(\varphi')$.
- If $\varphi' = \rho \texttt{ FILTER } P$, then $\texttt{STRICT}(\varphi') \equiv \texttt{STRICT}(\rho) \texttt{ FILTER } P$. By induction hypothesis, $\texttt{STRICT}(\rho)$ has an equivalent formula σ in S-CEL$\,\cup\{\;:\;, \oplus\}(\mathcal{P})$. Thus, $\psi' = \sigma \texttt{ FILTER } P$ is equivalent to $\texttt{STRICT}(\varphi')$.

- If $\varphi' = \rho_1$ `OR` ρ_2, then `STRICT`$(\varphi') \equiv$ `STRICT`(ρ_1) `OR` `STRICT`(ρ_2). By induction hypothesis, both `STRICT`(ρ_1) and `STRICT`(ρ_2) have equivalent formulas σ_1 and σ_2, respectively, in S-CEL$\,\cup\{:,\oplus\}(\mathcal{P})$. Thus, $\psi' = \sigma_1$ `OR` σ_2 is equivalent to `STRICT`(φ').
- If $\varphi' = \rho+$, then `STRICT`$(\varphi') \equiv$ `STRICT`$(\rho)\oplus$. By induction hypothesis, `STRICT`(ρ) has an equivalent formula σ in S-CEL$\,\cup\{:,\oplus\}(\mathcal{P})$. Thus, $\psi' = \sigma\oplus$ is equivalent to `STRICT`(φ').

After this the only thing left is to replace every subformula `STRICT`(φ') of φ with its S-CEL$\,\cup\{:,\oplus\}(\mathcal{P})$ equivalent ψ', and the resulting formula will be in S-CEL$\,\cup\{:,\oplus\}(\mathcal{P})$ and will be equivalent to φ, thus proving the lemma.

C.4 Proof of Theorem 16

C.4.1 S-CEL $\cup\{:\} \subseteq$ S-CEL $\cup\{$`STRICT`$\}$

For the first part we prove that for every formula φ in S-CEL$\,\cup\{:\}(\mathcal{P})$ there is a formula ψ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ such that $\varphi \equiv \psi$. For notation, for any formula ρ and labels A, B we write $\rho^{A\to B}$ to refer to the formula ρ after replacing every occurrence of A by B.

Consider a formula φ in S-CEL$\,\cup\{:\}(\mathcal{P})$. We are going to make use of a rather useful property of S-CEL$\{:\}(\mathcal{P})$, and of S-CEL in general, that is the fact that one can always push the labelling down to the assignments (or an assignment with filters). This is because any formula of the form $\varphi = \varphi'$ `IN` A can be rewritten as a new formula ψ that labels A one level lower than φ. This is done in the following way:

- If $\varphi' = \rho_1$ `OP` ρ_2 with `OP` $\in \{;\,,\,:\,,$ `OR` $\}$, then $\varphi \equiv (\rho_1$ `IN` $A)$ `OP` $(\rho_2$ `IN` $A)$.
- If $\varphi' = \rho$ `OP` with `OP` $\in \{+, \oplus\}$, then $\varphi \equiv (\rho$ `IN` $A)$ `OP`.
- If $\varphi' = \rho$ `FILTER` $P(\bar{A})$, then
  - if A is not in $\bar{A}$ then $\varphi \equiv (\rho$ `IN` $A)$ `FILTER` $P(\bar{A})$, and
  - if A is in $\bar{A}$ then $\varphi \equiv (\rho^{A\to A'}$ `IN` $A)$ `FILTER` $P(\bar{A}')$, where A' is a new label and $\bar{A}'$ is $\bar{A}$ replacing A with A'.

By using these equivalences, we can push down all labellings.

Now we prove that for every formula $\varphi' = \varphi_1 : \varphi_2$ with φ_1 and φ_2 in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ there exists a formula ψ' in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ equivalent to φ'. Here we assume that φ' is such that all its labels are applied at the lower level. Then the proof follows by doing induction over the structure of φ'. The base case is $\varphi' = R : T$ for some R, T. Clearly φ' is equal to $\psi' =$ `STRICT`$(R\,;T)$. The same works if φ_1 and φ_2 have the form $(R$ `IN` $A_1 \ldots$ `IN` $A_k)$. For this case, φ' would be equal to $\psi' =$ `STRICT`$(\varphi_1\,;\varphi_2)$. For the inductive step consider the following cases:

- If $\varphi_1 = \rho_1\,;\rho_2$, then $\varphi' \equiv \rho_1\,;(\rho_2 : \varphi_2)$ and $\rho_2 : \varphi_2$ is smaller than $\varphi_1 : \varphi_2$. By induction hypothesis, $(\rho_2 : \varphi_2)$ has an equivalent formula σ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$. Thus, $\psi' = \rho_1\,;\sigma$ is equivalent to φ'.
- If $\varphi_1 = \rho$ `FILTER` $P(\bar{A})$, then $\varphi' \equiv (\rho^{\bar{A}\to\bar{A}'} : \varphi_2)$ `FILTER` $P(\bar{A}')$, where $\bar{A}' = (A_1', \ldots, A_k')$ is a tuple of new labels with the same arity as $\bar{A} = (A_1, \ldots, A_k)$. By induction hypothesis, $(\rho^{\bar{A}\to\bar{A}'} : \varphi_2)$ has an equivalent formula σ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$. Thus, $\psi' = (\sigma$ `FILTER` $P(\bar{A}')$ is equivalent to φ'. Note that, since we renamed the labels $\bar{A}$ with $\bar{A}'$ in ρ, then for any filter $P'(\bar{B})$ with some $A_i \in B$ that is applied in a higher level, we must also add the filter $P(\bar{B}')$ where B' is B replacing A_i with A_i'.
- If $\varphi_1 = \rho_1$ `OR` ρ_2, then $\varphi' \equiv (\rho_1 : \varphi_2)$ `OR` $(\rho_2 : \varphi_2)$. By induction hypothesis, both $(\rho_1 : \varphi_2)$ and $(\rho_2 : \varphi_2)$ have equivalent formulas σ_1 and σ_2, respectively, in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$.

Thus, $\psi' = \sigma_1$ `OR` σ_2 is equivalent to φ'.

- If $\varphi_1 = \rho+$, then $\varphi' \equiv (\rho : \varphi_2)$ `OR` $(\rho+ \,;\, (\rho : \varphi_2))$. By induction hypothesis, $(\rho : \varphi_2)$ has an equivalent formula σ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$. Thus, $\psi' = \sigma$ `OR` $(\rho+ \,;\, \sigma)$ is equivalent to φ'.

The cases regarding the structure of φ_2 instead of φ_1 are analogous to the previous ones.

Then, every subformula φ' of φ is replaced by its equivalent ψ' in a bottom-up fashion to ensure that the subformulas of φ' are indeed in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$. Finally, the resulting formula ψ is in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ and is equivalent to φ.

C.4.2 S-CEL $\cup\{\oplus\} \not\subseteq$ S-CEL $\cup\{$`STRICT`$\}$

Now, for the second part we prove that there is a set $\mathcal{P}$ containing a single binary set predicate and a formula $\varphi \in$ S-CEL$\,\cup\{\oplus\}(\mathcal{P})$ that is not equivalent to any formula in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$. In particular, consider $\mathcal{P} = \{P^{\mathrm{S}}_{=}\}$, where $P_{=}(x, y) := (x.a = y.a)$, and consider the formula:

$$\varphi \;=\; ((A\,;\,E)\ \texttt{FILTER}\ P^{\mathrm{S}}_{=}(A, E))\oplus$$

in S-CEL$\,\cup\{\oplus\}(\mathcal{P})$. We prove that there is no formula ψ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ equivalent to φ. For this we first give a few somehow useful definitions.

Consider a stream S, a complex event C, a valuation μ, two positions $i, j \in C$ with $i < j$ and a constant $k \geq 1$. Consider the factorization $C_1 \cdot C_2 \cdot C_3$ of C given by i, j in which C_1 contains all positions in C lower than i, C_2 contains all positions of C between i and j (including them) and C_3 contains all positions of C higher than j. Likewise, consider the factorization $S_1 \cdot S_2 \cdot S_3$ and $\mu_1 \cup \mu_2 \cup \mu_3$ of S and μ in the same way.

Now, we define the result of pumping the fragment $[i, j]$ of (S, C, μ) k-times as a tuple (S', C', μ'), where S', C' and μ' are a stream, complex event and valuation, defined as follows:

- To define S', we consider two cases. First, when C_2 does not induce a contiguous interval (that is, there is some l such that $i < l < j$ and $l \notin C_2$), we define S' as $S_1 \cdot P_0 \cdot S_2 \cdot P_1 \cdot S_2 \cdot \ldots \cdot S_2 \cdot P_k \cdot S_3$, where S_2 is repeated k times and each P_i is an arbitrary finite stream. Second, when C_2 does induce a contiguous interval, we define S' without the P_i, i.e., $S' = S_1 \cdot S_2 \cdot S_2 \cdot \ldots \cdot S_2 \cdot S_3$.
- C' is defined as $C_1 \cdot C_2^1 \cdot C_2^2 \cdot \ldots \cdot C_2^k \cdot C_3'$, where each C_2^i is the same complex event C_2 but with its values moved to fit the i-th occurrence of S_2 in S'. For example, C_2^2 results after adding $|S_2|$ to all values of C_2 if it induces a contiguous interval, and adding $|P_0| + |S_2| + |P_1|$ else. Likewise, C_3' is the same as C_3 but moved to fit S_3.
- Like for C', μ' is defined as $\mu_1 \cup \mu_2^1 \cup \mu_2^2 \cup \ldots \cup \mu_2^k \cup \mu_3'$, where each μ_2^i is the same valuation μ_2 but with its values moved to fit the i-th occurrence of S_2 in S'. Likewise, μ_3' is the same as μ_3 but moved to fit S_3.

Notice that if C induces a contiguous interval, then also does C'. Moreover, notice that no new events were added to the valuation, i.e. $S[\mu(A)] = S'[\mu'(A)]$ for every label A.

A formula ρ in S-CEL is said to be *pumpable* if there exists a constant $N \in \mathbb{N}$ such that for every stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\rho]\!](S, p_1, p_2, \mu)$ with $|C| > N$ there exist two positions $i, j \in C$ with $i < j$ such that for every $k \geq 1$ it holds that $C' \in [\![\rho]\!](S', p_1, p_2', \mu')$, where (S', C', μ') is the results of pumping the fragment $[i, j]$ of (S, C, μ) k-times and p_2' is the position at which $S[p_2]$ ended. In the following lemma we show the utility of this property.

▶ **Lemma 19.** *Every formula φ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ is pumpable.*

Proof. Consider a formula φ in S-CEL$\cup\{$`STRICT`$\}(\mathcal{P})$. We prove the lemma by induction over the structure of φ. First, consider the base case R. By defining $N = 1$ we know that for every stream S there is no complex event $C \in [\![\varphi]\!](S)$ with $|C| > N$, so the lemma holds.

Now, for the inductive step consider first the case $\varphi = \psi_1$ `FILTER` $P(X_1, \ldots, X_n)$. By induction hypothesis, we know that the lemma holds for ψ_1, thus let N_1 be its corresponding constant. Let N be equal to N_1. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. By definition $C \in [\![\psi_1]\!](S, p_1, p_2, \mu)$ and $(S[\mu(X_1)], \ldots, S[\mu(X_n)]) \in P$. By induction hypothesis, ψ_1 is pumpable, thus there exist positions $i, j \in C$ with $i < j$ such that the fragment $[i, j]$ can be pumped. Moreover, consider that the result of pumping the fragment $[i, j]$ k times is (S', C', μ'), for an arbitrary k. Then, it holds that $C' \in [\![\psi_1]\!](S' p_1, p_2', \mu')$. Also, because in the pumping it holds that $S[\mu(A)] = S'[\mu'(A)]$ for every A, then $(S'[\mu'(X_1)], \ldots, S'[\mu'(X_n)]) \in P$. Therefore, $C' \in [\![\varphi]\!](S' p_1, p_2', \mu')$, thus φ is pumpable.

Consider now the case $\varphi = \psi_1$ `OR` ψ_2. By induction hypothesis, we know that the property holds for ψ_1 and ψ_2, thus let N_1 and N_2 be the corresponding constants, respectively. Then, we define the constant N as the maximum between N_1 and N_2. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. By definition or `OR`, either $C \in [\![\psi_1]\!](S, p_1, p_2, \mu)$ or $C \in [\![\psi_2]\!](S, p_1, p_2, \mu)$, so w.l.o.g. consider the former case. By induction hypothesis, ψ_1 is pumpable, thus there exist positions $i, j \in C$ with $i < j$ such that the fragment $[i, j]$ can be pumped and the result (S', C', μ') satisfies $C' \in [\![\psi_1]\!](S', p_1, p_2', \mu')$. This means that $C' \in [\![\varphi]\!](S', p_1, p_2', \mu')$, therefore, φ is pumpable.

Now, consider the case $\varphi = \psi_1 \,;\, \psi_2$. By induction hypothesis, we know that the property holds for ψ_1 and ψ_2, thus let N_1 and N_2 be the corresponding constants, respectively. Then, we define the constant $N = N_1 + N_2$. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. This means that there exist $p' \in \mathbb{N}$, complex events C_1, C_2 and valuations μ_1, μ_2 such that $C = C_1 \cdot C_2$, $\mu = \mu_1 \cup \mu_2$, $C_1 \in [\![\psi_1]\!](S, p_1, p', \mu_1)$ and $C_2 \in [\![\psi_2]\!](S, p' + 1, p_2, \mu_2)$. Moreover, either $|C_1| > N_1$ or $|C_2| > N_2$, so w.l.o.g. assume the former case. By induction hypothesis, ψ_1 is pumpable, thus there exist positions $i, j \in C_1$ with $i < j$ such that the fragment $[i, j]$ can be pumped and the result (S', C_1', μ_1') satisfies $C_1' \in [\![\psi_1]\!](S', p_1, p' + r, \mu_1')$, assuming that the pumping added r new events. Define the complex event $C' = C_1' \cdot C_2'$, where C_2' is the same as C_2 but adding r to each position (so that $(S[C_2] = S'[C_2'])$. Similarly, define the valuation $\mu' = \mu_1' \cup \mu_2'$, where μ_2' is μ_2 adding r to each value. Then $C_1' \in [\![\psi_1]\!](S', p_1, p' + r, \mu_1')$ and $C_2' \in [\![\psi_2]\!](S', p' + r + 1, p_2 + r, \mu_2')$, thus $C' \in [\![\varphi]\!](S', p_1, p_2 + r, \mu')$, therefore, φ is pumpable.

Consider then the case $\varphi = \psi_1 +$. By induction hypothesis, we know that the lemma holds for ψ_1, thus let N_1 be its corresponding constant. Let the constant N be equal to N_1. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. Then, consider $i = \min(C)$ and $j = \max(C)$, consider any $k \geq 1$ and let (S', C', μ') be the result of pumping the fragment $[i, j]$ of (S, C, μ) k times. We prove now that $C' \in [\![\varphi]\!](S', p_1, p_2', \mu')$ by induction over k. If $k = 1$ then, as defined in the definition of pumping, S' has the form $S_1 \cdot P_0 \cdot S_2 \cdot P_1 \cdot S_3$, and C' and μ' are the same as C and μ but adding r to each position, where $r = |P_0|$. Clearly it holds that $C' \in [\![\varphi]\!](S', p_1, p_2', \mu')$, since the modifications did not affect the complex event part of S. Now, consider that $k > 1$. Then, S' has the form $S_1 \cdot P_0 \cdot S_2 \cdot P_1 \cdot S_2 \cdot \ldots \cdot S_2 \cdot P_k \cdot S_3$. Similarly, C' and μ' are defined as $C_1 \cdot C_2^1 \cdot C_2^2 \cdot \ldots \cdot C_2^k \cdot C_3'$ and $\mu_1 \cup \mu_2^1 \cup \mu_2^2 \cup \ldots \cup \mu_2^k \cup \mu_3'$, respectively, where $C_1 = C_3' = \emptyset$, $\mu_1(A) = \mu_3(A) = \emptyset$ for any A, and each C_2^i and μ_2^i are the same complex event C and valuation μ but with their positions moved to fit the i-th occurrence of S_2 in S'. Consider that $r = |S_1 \cdot P_0 \cdot S_2|$. By induction hypothesis, we can say that $C_2' \in [\![\varphi]\!](S', r + 1, p_2', \mu_2')$

where $C_2' = C_2^2 \cdot \ldots \cdot C_2^k$ and $\mu_2' = \mu_2^2 \cup \ldots \cup \mu_2^k$ (notice that we consider it from position $r+1$ because there is no lower position in the complex event). Also, it is easy to see that this implies $C_2' \in [\![\varphi{+}]\!](S', r+1, p_2', \mu_2')$, which is something we will need next. Moreover, it holds that $C_2^1 \in [\![\varphi]\!](S', p_1, r, \mu_2^1)$, because it represents the same complex event as the original one C. Then, because $C' = C_2^1 \cdot C_2'$ and $\mu' = \mu_2^1 \cup \mu_2'$, it follows that $C' \in [\![\varphi\,;\,\varphi{+}]\!](S', p_1, p_2', \mu')$ which also implies that $C' \in [\![\varphi{+}]\!](S', p_1, p_2', \mu')$. Since $\varphi{+} = \psi_1{+}{+} \equiv \psi_1{+} = \varphi$, it holds that $C' \in [\![\varphi]\!](S', p_1, p_2', \mu')$.

Now, consider the case $\varphi = \texttt{STRICT}(\psi_1)$. By induction hypothesis, we know that the lemma holds for ψ_1, thus let N_1 be its corresponding constant. Let the constant N for φ be equal to N_1. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. Then, by definition $C \in [\![\psi_1]\!](S, p_1, p_2, \mu)$, and by induction hypothesis there exist positions $i, j \in C$ such that the fragment $[i, j]$ can be pumped and the result (S', C', μ') satisfies $C' \in [\![\psi_1]\!](S, p_1, p_2', \mu')$. Notice that C induces a contiguous interval because of the definition of `STRICT`, therefore C' also induces a contiguous interval, thus $C' \in [\![\varphi]\!](S, p_1, p_2', \mu')$.

Finally, consider the case $\varphi = \psi_1$ `IN` A. By induction hypothesis, we know that the lemma holds for ψ_1, thus let N_1 be its corresponding constant. Let the constant N be equal to N_1. Consider any stream S, valuation μ, positions p_1, p_2 and complex event $C \in [\![\varphi]\!](S, p_1, p_2, \mu)$ with $|C| > N$. Then, by definition $\mu(A) = C$, and there exists η such that $C \in [\![\psi_1]\!](S, p_1, p_2, \eta)$ and $\mu(B) = \eta(B)$ for all $B \neq A$. By induction hypothesis there exist positions $i, j \in C$ such that the fragment $[i, j]$ can be pumped and the result (S', C', η') satisfies $C' \in [\![\psi_1]\!](S, p_1, p_2', \eta')$. Note that the results (S', C', μ') and (S, C', η') of pumping $[i, j]$ in (S, C, μ) and (S, C, η), respectively, are the same, with the only difference that μ' satisfies $\mu'(A) = C'$. Then it follows that $C' \in [\![\varphi]\!](S, p_1, p_2', \mu')$. ◀

Now, we show that there is no formula ψ in S-CEL$\,\cup\{$`STRICT`$\}(\mathcal{P})$ equivalent to $\varphi = ((A\,;\,E)$ `FILTER` $P_{=}^{\mathrm{S}}(A, E))\oplus$ by proving that such formula is not pumpable. By contradiction, assume that ψ exists, and let N be its constant. Consider then the stream:

$$S = \begin{array}{cccccccccccc} A & L & E & A & L & E & \cdots & A & L & E & \cdots \\ 1 & 1 & 1 & 2 & 2 & 2 & & N & N & N & \end{array}$$

Where the first and second lines are the type and a attribute of each event, respectively, and consider the complex event $C = \{1, 3, 4, 6, \ldots, 3N-2, 3N\}$ and valuation μ with $\mu(A) = \{1, 4, 7, \ldots, 3N-2\}$ and $\mu(E) = \{3, 6, 9, \ldots, 3N\}$. Now, let $i, j \in C$ be any two positions of the complex event, which define the partitions $C_1 \cdot C_2 \cdot C_3$ and $\mu_1 \cup \mu_2 \cup \mu_3$, and name $t_1 = S[i]$ and $t_2 = S[j]$. We will use $k = 2$, i.e., repeat section $S[i, j]$ two times, and use the 1-tuple stream $U(0)$ as the arbitrary streams P_0 and P_1 to get the resulting stream S' and the corresponding complex event C' and valuation μ'. We will analyse the following possible cases: $\text{type}(t_1) = \text{type}(t_2)$; $\text{type}(t_1) = A$ and $\text{type}(t_2) = E$; $\text{type}(t_1) = E$ and $\text{type}(t_2) = A$. In the first case the resulting C' is a complex event with two consecutive tuples of the same type, which contradicts the original formula φ. In the second case C_2 is not a contiguous interval so the complex event C' would fail to ensure that the A tuple following t_2 is placed right after it (because of the tuple $U(0)$ inbetween), thus contradicting the $\oplus$ property of φ. In the third case it is clear that the last A in the first repetition of $[i, j]$ and the first E in the second repetition (i.e., $S[j]$ and $S[j+2]$) do not satisfy the `FILTER` condition because $S[j].a > S[j+2].a$. Finally, the formula ψ cannot exist.