On the Expressiveness of Languages for Complex Event Recognition

Alejandro Grez
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
ajgrez@uc.cl

Cristian Riveros
Pontificia Universidad Católica de Chile, Santiago, Chile
Millennium Institute for Foundational Research on Data, Santiago, Chile
cristian.riveros@uc.cl

Martín Ugarte
Millennium Institute for Foundational Research on Data, Santiago, Chile
martin.ugarte@imfd.cl

Stijn Vansummeren
Université Libre de Bruxelles, Brussels, Belgium
stijn.vansummeren@ulb.ac.be

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 these languages are naturally first-order, meaning that variables 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-Oriented Complex Event Logic (SO-CEL for short), a variation of the CER language introduced in [17] in which all variables bind to sets of matched events. We then compare SO-CEL with CEL, the CER language of [17] 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 SO-CEL is strictly more expressive than CEL. To get a better understanding of the expressive power, computational capabilities, and limitations of SO-CEL, we also investigate the relationship between SO-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, SO-CEL captures precisely the subclass of CEA that satisfy this property. Finally, we identify the operations that SO-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.2020.15

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

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.

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 [10, 9]; human activities in video content [18]; traffic incidents in smart cities [4]; and opportunities in the stock market [20]. Numerous systems for processing streaming data have been proposed over the decades (see, e.g., [19, 12] for surveys). Complex Event Recognition (CER for short) systems are specialized 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, numerous CER systems and languages have been proposed in the literature – see e.g., the surveys [12, 3] 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 [17]. This logic, denoted CEL, combines the regular expression operators (sequencing, to require that some pattern occurs before another somewhere in a stream; iteration, to recognize a pattern a number of times; and disjunction) with data filtering features as well as limited data outputting capabilities. CEL follows the approach that seems to be taken by most of the CER literature (e.g., [14, 15, 1, 24, 11], see also [19, 12]) 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. In this respect, variables in CEL are *first order* variables, since they bind to individual events.

One of the main contributions of 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, first order variables interact rather awkwardly with Kleene closure. Indeed, if a variable is bound inside Kleene closure, what does the variable refer to when used outside of a Kleene closure? In many of the practical CER languages, first order variables are used inside Kleene Closure to express properties on sequences of events rather than on individual events. In other words, first order variables are used to express properties on *second order* objects.

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 freezing plantations events 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 first order variables and second order properties 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 [15, 14, 13] and SASE [24, 1, 23]. 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:

$$\texttt{FILTER}\{value < 0\}\ T$$
$$\texttt{FOLD}\big\{\$2.value < \$.value,\ \$2.value \geq \$.value\ \texttt{AND}\ \$2.value \geq 60\big\}\ H \quad (1)$$

   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. $\$2.value \geq \$.value$ AND $\$2.value \geq 60$).
2. In SASE, we can define the complex event more directly with the following query:

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

   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 first- and second-order objects are mixed. Cayuga uses first order variables (e.g. $ and $2) to define, in a procedural way, a property over a second order object: the set of H events. Instead, the SASE query implicitly combines first order variables (e.g. t and $h2$) with second order variables (e.g. $h1[]$). Indeed, SASE uses the ad-hoc notation $h1[i-1].value < h1[i].value$ to declare a predicate over the second order object $h1[]$. It is important to mention that in both languages the semantics of the second order objects is not formally defined and,

[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.

moreover, second order objects are actually not acknowledged as such. As a consequence, CER languages are designed without any understanding of what the expressive power is of using first order versus second order 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 exact relationship in expressive power between the CER language and their computational models has never been studied before.

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 CEL [17] as a model of the former class of languages, and we introduce Set Oriented Complex Event Logic (SO-CEL for short) as a model for the second class of languages. Variables in SO-CEL can only bind and filter sets of matched events. Specifically, we compare CEL against SO-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, SO-CEL is strictly more expressive than CEL. However, when equipped with sets of ternary predicates, the languages are incomparable. The intuition behind this is that SO-CEL cannot distinguish the events captured by a single variable inside a Kleene closure, while this is possible in CEL by using a clever trick that relies on ternary filters (Section 4).

Since CEL and SO-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* [17] (CEA for short). We show that, in this setting, CEL and SO-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 CEL and SO-CEL while others, such as `UNLESS`, provide the languages with new capabilities (Section 5).

Finally, we identify the operations that SO-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 little formal study of the expressiveness of CER languages. A notable exception is the work by Zhang et al on SASE$^+$ [24], which considers the descriptive complexity of a core CER language. It is unfortunate, however, that this paper lacks a formal definition of the language under study; and ignores in particular the aforementioned issues related to the scoping of variables under Kleene closure, as well as the data output capabilities.

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 SO-CEL and 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 [16]. 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 second-order variables, such as monadic second order logic (MSO), are standard in logic and databases [21]. However, to the best of our knowledge we are not aware of any CER language such as SO-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 CEL, as introduced in [17].

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 $S = t_0 t_1 \dots$ 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 $S = t_0 t_1 \dots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$, and the sub-stream $t_i t_{i+1} \dots$ 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 [22]) to future work.

CEL syntax. Let $\mathbf{X}$ be a set of first order variables. Given a schema $\mathcal{R}$, an FO 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 FO predicates then an atom over $\mathcal{P}$ is an expression $P(x_1, \dots, x_n)$ with $P \in \mathcal{P}$ of arity n and $x_1, \dots, x_n$ FO variables in $\mathbf{X}$. The set of formulas of 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 $\mathcal{P}$.

CEL semantics. For the semantics of 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 CEL formula φ, we denote by $\text{vdef}(\varphi)$ all variables defined in φ by a clause of the form $R \;\texttt{AS}\; x$ 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 $\text{vdef}_+(\varphi) = \{x\}$. 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 CEL. Given an 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 $i \leq j$ and one of the following conditions holds:

- $\varphi = R$ `AS` x, $C = \{\nu(x)\}$, $\text{type}(S[\nu(x)]) = R$ and $i \leq \nu(x) = j$.[2]
- $\varphi = \rho$ `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 $C \in [\![\rho]\!](S, i, j, \nu[\nu'/U])$ 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 CEL in [17] did not use the bounds i and j. We use them here just for consistency with the other definitions in the paper (SO-CEL and SO-CEL+).

▶ **Example 1.** Consider that we want to use 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 FO 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 Second-Order Complex Event Logic

In this section, we formally define SO-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 SO-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 SO-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*

[2] The fact that $\nu(x) = j$ implies that the event in position j will always be part of the matched complex event. The reason behind this design decision is that one does not desire to wait for future events to decide whether or not a complex event matches a certain pattern.

a temperature below 0 degrees, there is a period where humidity increases until humidity is over 60%". This statement can be defined in SO-CEL with the following formula:

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

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 SO-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 SO-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 SO-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 SO-CEL.

SO-CEL syntax. Let $\mathbf{L}$ be a finite set of SO (Set-Oriented) variables containing all relation names (i.e. $\mathcal{R} \subseteq \mathbf{L}$). An SO predicate of arity n is an n-ary relation P over sets of tuples, $P \subseteq (2^{\mathrm{tuples}(\mathcal{R})})^n$. We write $\mathrm{arity}(P)$ for the arity of P. Let $\mathcal{P}$ be a set of SO 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 SO-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 $\mathcal{P}$.

SO-CEL semantics. An SO 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 $\mathrm{supp}(\mu) = \bigcup_{a \in \mathbf{L}} \mu(A)$. 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 SO-CEL formulas. Given a SO-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 SO valuation μ (denoted by $C \in [\![\varphi]\!](S, i, j, \mu)$) if one of the following conditions holds:

- $\varphi = R$, $C = \mu(R) = \{j\}$, $\text{type}(S[j]) = R$ and $\mu(A) = \emptyset$ for every $A \neq R$.
- $\varphi = \rho$ `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 $B \neq A$.
- $\varphi = \rho$ `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.
- $\varphi = \rho_1$ `OR` ρ_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 CEL we say that C belongs to the evaluation of a SO-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 SO 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 SO-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 SO-CEL more flexible and simpler than the one of 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 first-order variables in order to manage second-order objects (i.e. complex events). For example, in CEL variables can only be assigned at the event definition, with the atomic formula R `AS` x. In contrast, second-order variables can manage complex events, allowing to use the `IN`-operator anywhere in a formula. Another more interesting example is iteration. In order to use first-order 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, SO 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 first-order and second-order variables. Given that in this paper our expressiveness analysis is always between CEL and SO-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 The Expressiveness of SO variables versus FO variables

In this section, we compare the expressiveness of CEL and SO-CEL. Since in traditional logics second-order languages can encode everything a first-order language can, this could suggest that SO-CEL is more expressive than CEL. We show that this is only partially true: SO-CEL includes CEL for binary predicates but they are incomparable in general.

In order to make a fair comparison between CEL and SO-CEL we first need to agree on how we relate the FO predicates that can be used in CEL to the SO predicates that can be used in SO-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 SO-CEL we can easily express formulas that are beyond the scope of CEL. For this reason, we will restrict ourselves to SO predicates created as *extensions* of FO predicates. Given an FO predicate $P(x_1, \ldots, x_n)$, we define its *SO-extension* P^{SO} to be the SO predicate of the same arity as P such that $(S_1, \ldots, S_n) \in P^{\text{SO}}$ 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 FO predicates, $\mathcal{P}^{\text{SO}}$ is the set $\{P^{\text{SO}} \mid P \in \mathcal{P}\}$. In what follows we will compare CEL($\mathcal{P}$) to SO-CEL($\mathcal{P}^{\text{SO}}$).

▶ **Example 3.** Using the SO-extensions of the unary FO 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 CEL expression of Example 1 can be written in SO-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 FO 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, SO-CEL cannot meaningfully filter events captured by a Kleene closure. For example, if $X.\texttt{id} = Y.\texttt{id}$ 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 CEL with SO-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 FO predicates, CEL($\mathcal{U}$) and SO-CEL($\mathcal{U}^{\text{SO}}$) 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 FO predicates. For every formula $\psi \in$ CEL($\mathcal{U}$) there exists a formula $\varphi \in$ SO-CEL($\mathcal{U}^{\text{SO}}$) 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. CEL with unary predicates was extensively studied in [17] 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 SO-CEL formulas that cannot be equivalently defined in CEL (under the same set of predicates). Let $\mathcal{P}_=$ be the smallest set of FO predicates that allows to express equality between attributes of tuples and is closed under boolean operations.

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

An example of a formula that can be defined in SO-CEL($\mathcal{P}_=^{\text{SO}}$) but cannot be defined in CEL($\mathcal{P}_=$) is $\varphi := (R+;T+)$ `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 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 comparisons in the evaluation of an CEL formula is at most linear in the size of the output and, thus, φ cannot be defined by any formula in CEL($\mathcal{P}_=$). It is important to note that this result shows the limitations of a CER language based on FO variables and what can be gained if SO variables are used.

A natural question at this point is whether SO-CEL can define every 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 FO binary predicates closed under complement. Then for every formula $\psi \in$ CEL($\mathcal{B}$) there exists a formula $\varphi \in$ SO-CEL($\mathcal{B}^{\text{SO}}$) 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 SO-CEL is always more expressive than CEL. In particular, there exists an CEL formula with ternary predicates that cannot be defined by any SO-CEL formula. For the next result, consider the smallest set of FO predicates $\mathcal{P}_+$ containing the sum predicate $x = y + z$ that is closed under boolean operations.

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

The formula R `AS` x; (S `AS` y; T `AS` z `FILTER` $(x = y + z)$)+ cannot be defined in SO-CEL($\mathcal{P}_+^{\text{SO}}$). 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 SO-CEL given that in SO-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 CEL($\mathcal{P}$) or SO-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 FO predicates. As we showed in Section 4, CEL($\mathcal{U}$) and SO-CEL($\mathcal{U}^{\text{SO}}$) are equally expressive in this setting, suggesting that this is a robust subfragment of CER query languages. In this section, we compare CEL and SO-CEL with complex event automata (CEA), a computational model proposed in [17] for efficiently evaluating CEL with unary FO predicates. We show that the so-called $*$-property of CEA captures the expressiveness of CEL and SO-CEL with unary predicates. Furthermore, we use this property to understand the expressiveness of CEL and SO-CEL under the extension with new CER operators.

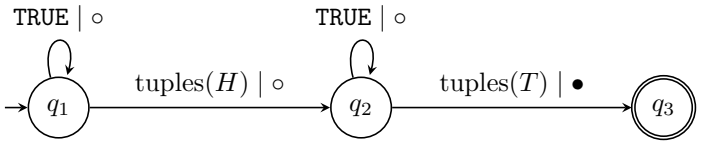

Figure 2 A complex event automaton that has no equivalent formula in SO-CEL.

Let $\mathcal{R}$ be a schema and $\mathcal{U}$ be a set of unary FO 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* [17] (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 $C_\rho = \{i \in [0, n] \mid m_i = \bullet\}$. 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 [17], it was shown that for every formula $\varphi \in \text{CEL}(\mathcal{U})$ 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 $\varphi \in \text{SO-CEL}(\mathcal{U}^{\text{SO}})$ 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 $\text{SO-CEL}(\mathcal{U}^{\text{SO}})$ (and thus in $\text{CEL}(\mathcal{U})$). 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 SO-CEL formulas because such a formula would either not check that the H event occurs, or include the position j of H in C – which the automaton does not.

In order to capture the exact expressiveness of CEL or SO-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 FO predicates, SO-CEL($\mathcal{U}^{\text{SO}}$) has the same expressive power as CEA($\mathcal{U}$) under the $*$-semantics, namely, for every formula φ in SO-CEL($\mathcal{U}^{\text{SO}}$), 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 $f : \text{streams}(\mathcal{R}) \to 2^{\mathbf{C}}$, where $\text{streams}(\mathcal{R})$ 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 SO-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 SO-CEL($\mathcal{U}^{\text{SO}}$) such that $f = [\![\varphi]\!]$.*

With the previous corollary we have captured the exact expressiveness of CEL($\mathcal{U}$) and SO-CEL($\mathcal{U}^{\text{SO}}$) 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 SO-CEL($\mathcal{U}^{\text{SO}}$). Some languages include additional useful operators like `AND`, `ALL` and `UNLESS`, which have the following semantics in SO-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 FO predicates. For every expression φ of the form φ_1 OP φ_2, with OP $\in \{\texttt{AND}, \texttt{ALL}\}$ and φ_i in SO-CEL($\mathcal{U}^{\text{SO}}$), there is a SO-CEL($\mathcal{U}^{\text{SO}}$) 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 SO-CEL($\mathcal{U}^{\text{SO}}$) with $\mathcal{U}$ unary FO predicates. This shows that UNLESS adds expressibility to unary SO-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 FO predicates, SO-CEL($\mathcal{U}^{\text{SO}}$) captures the class of CEA over $\mathcal{U}$ that have the $*$-property (Corollary 10). However, in [17] 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 SO-CEL

By looking at the characterization of SO-CEL in terms of the $*$-property, one can easily distinguish three shortcomings of SO-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 second-order variables it is straightforward to introduce a simple projection operator in SO-CEL.

The second shortcoming of SO-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 [23] 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 SO-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 SO-CEL with the following operators:

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

where $L \subseteq \mathbf{L}$. Recall that for a valuation μ, $\text{supp}(\mu)$ is defined as $\text{supp}(\mu) = \bigcup_{A\in\mathbf{L}} \mu(A)$. 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 $C \in \bigcup_{k=1}^{\infty} [\![\rho^k]\!](S, i, j, \mu)$ where $\rho^k = \rho : \cdots : \rho$ k-times.
- $\varphi = \pi_L(\rho)$, $C = \text{supp}(\mu)$ 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.
- $\varphi =$ $\texttt{START}(\rho)$, $C \in [\![\varphi']\!](S, i, j, \mu)$, and $\min(C) = i$.

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

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:

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

Having defined the previous operators, we proceed to show that for every set $\mathcal{U}$ of unary predicates, SO-CEL+ ($\mathcal{U}^{\text{SO}}$) captures the full expressive power of CEA over $\mathcal{U}$. To this end, we say that a formula φ in SO-CEL+ ($\mathcal{U}^{\text{SO}}$) 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 FO predicates. For every CEA $\mathcal{A}$ over $\mathcal{U}$, there is a formula $\varphi \in$ SO-CEL+ ($\mathcal{U}^{\text{SO}}$) such that $\varphi \equiv \mathcal{A}$. Conversely, for every formula $\varphi \in$ SO-CEL+ ($\mathcal{U}^{\text{SO}}$) there exists a CEA $\mathcal{A}$ over $\mathcal{U}$ such that $\varphi \equiv \mathcal{A}$.*

This result is particularly relevant because, as shown in [17], 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 SO-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 [23] 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 SO-CEL let $\texttt{STRICT}(\varphi)$ 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 $C \in [\![\texttt{STRICT}(\varphi)]\!](S, i, j, \mu)$ 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 seem equally expressive since each allows to force contiguity between pairs of events. At least, this intuition holds whenever we restrict to unary predicates.

▶ **Proposition 14.** *Let $\mathcal{U}$ be a set of unary SO predicates. For every φ in SO-CEL$\cup\{:, \oplus\}(\mathcal{U})$, there exists a formula ψ in SO-CEL$\cup\{\texttt{STRICT}\}(\mathcal{U})$ 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 SO predicates. Given a formula $\varphi \in$ SO-CEL$\cup\{\texttt{STRICT}\}(\mathcal{P})$ there exists $\psi \in$ SO-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 SO predicates and for any formula $\varphi \in$ SO-CEL$\cup\{:\}(\mathcal{P})$ there is a formula $\psi \in$ SO-CEL$\cup\{\texttt{STRICT}\}(\mathcal{P})$ such that $\varphi \equiv \psi$. In contrast, there exists a set $\mathcal{P}$ containing a single binary SO predicate and a formula $\varphi \in$ SO-CEL$\cup\{\oplus\}(\mathcal{P})$ that is not equivalent to any formula in SO-CEL$\cup\{\texttt{STRICT}\}(\mathcal{P})$.*

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 SO-CEL and monadic second-order logic (MSO). For unary filters, we conjecture that SO-CEL+ has the same expressive power as MSO over unary filters. Another natural question is to compare the expressiveness of SO-CEL+ and MSO extended with binary predicates. Furthermore, a more fundamental question is what fragments of SO-CEL or MSO (with binary predicates) can be evaluated with strong guarantees like constant-delay enumeration. We believe that understanding the relation between SO-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. doi:10.1142/S012905410300214X.

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. doi:10.1007/978-3-642-00982-2_24.

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. 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 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. doi:10.1145/2699442.

17 Alejandro Grez, Cristian Riveros, and Martin Ugarte. A formal framework for Complex Event Processing. In *ICDT*, 2019.

18 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.

19 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. doi:10.1145/3299887.3299892.

20 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.

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

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

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

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