Tutorial: Complex Event Recognition Languages

Alexander Artikis
University of Piraeus
NCSR Demokritos
Athens, Greece
a.artikis@iit.demokritos.gr

Alessandro Margara
Politecnico di Milano
Milano, Italy
alessandro.margara@polimi.it

Martin Ugarte
Université Libre de Bruxelles
Brussels, Belgium
martin.ugarte@ulb.ac.be

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

Matthias Weidlich
Humboldt-Universtität zu Berlin
Berlin, Germany
matthias.weidlich@hu-berlin.de

ABSTRACT

Complex event recognition (CER) refers to the detection of events in Big Data streams. The paper presents a summary of the most prominent models and algorithms for CER, and discusses the main conceptual links and the differences between them.

ACM Reference format:
Alexander Artikis, Alessandro Margara, Martin Ugarte, Stijn Vansummeren, and Matthias Weidlich. 2017. Tutorial: Complex Event Recognition Languages. In *Proceedings of DEBS '17, Barcelona, Spain, June 19-23, 2017,* 5 pages.
DOI: http://dx.doi.org/10.1145/3093742.3095106

1 INTRODUCTION

In a broad range of domains, contemporary applications require processing of continuously flowing data from geographically distributed sources at extremely high and unpredictable rates to obtain timely responses to complex queries. Complex event recognition (CER) — event pattern matching — refers to the detection of events in Big Data streams, thereby providing the opportunity to implement reactive and proactive measures. Example applications consist of the recognition of human activities in video content, violations of maritime regulations and trading opportunities in the stock market. In each scenario, CER allows to make sense of streaming data, react accordingly and potentially prepare for taking counter-measures.

Numerous CER systems and languages have been proposed in the literature—see [5, 9] for two surveys. These systems have a common goal, but differ in their architectures, data models, pattern languages, and processing mechanisms. Their comparative assessment is further hindered by the fact that many of the techniques have been developed in different communities, each bringing in their own terminology and view on the problem. This is then the aim of our DEBS 2017 tutorial: to present a unified view of the foundations of CER, allowing for a *comparison* of different approaches. The focus is on the *formal* methods for CER as they have been developed in the database, distributed systems, and artificial intelligence communities. More precisely, we review models based on automata, tree structures, and logic-based rules. For each of these models, we compare the most important recognition algorithms, such as recognition based on the creation and storage of partial automata runs, the use of dynamic trees for storing intermediate matches, or inference for event models based on first-order logic. This paper presents a succinct summary of the different models that will be discussed during the tutorial, as well as some of the conceptual links between them. Because of space constraints, discussion is necessarily broad rather than deep.

Running Example. To facilitate the presentation of the reviewed approaches, we will use a simple example. Assume that we are interested in monitoring the temperature and humidity in some farm to detect various types of hazard, such as wood fires. To this effect, sensors are placed in designated areas within the farm, reporting temperature and humidity values. Each produced sensor event contains the area in which the sensor is located and a measurement value. Certain combinations of such sensor events indicate that with some probability there is a hazardous situation; for instance, wood fires are likely to occur in certain areas when humidity drops below 25% while temperature is higher than 40 degrees.

2 AUTOMATA-BASED MODELS

Many CER systems provide users with a pattern language that is later compiled into some form of automaton. The automaton model is generally used to provide the semantics of the language and/or as an execution framework for pattern recognition. Examples of automata-based CER systems include Cayuga [6], SASE [16] and SASE+ [17] — which all use automata for both purposes — and TESLA [7], which uses automata for pattern recognition. In this section, we first discuss the kinds of automata that are typically used in CER systems and then overview the recognition algorithms that have been proposed for them.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
DEBS '17, Barcelona, Spain
© 2017 Copyright held by the owner/author(s). 978-1-4503-5065-5/17/06...$15.00
DOI: http://dx.doi.org/10.1145/3093742.3095106

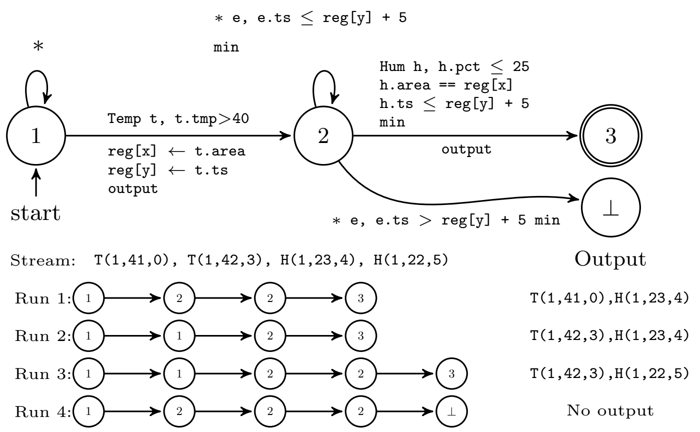
* e, e.ts ≤ reg[y] + 5
min
*
1
start
Temp t, t.tmp>40
reg[x] ← t.area
reg[y] ← t.ts
output
2
Hum h, h.pct ≤ 25
h.area == reg[x]
h.ts ≤ reg[y] + 5
min
output
3
⊥
* e, e.ts > reg[y] + 5 min
Stream: T(1,41,0), T(1,42,3), H(1,23,4), H(1,22,5)
Output
Run 1: 1 2 2 3 T(1,41,0),H(1,23,4)
Run 2: 1 1 2 3 T(1,42,3),H(1,23,4)
Run 3: 1 1 2 2 3 T(1,42,3),H(1,22,5)
Run 4: 1 2 2 2 ⊥ No output

Figure 1: Example CER automaton and runs over a stream.

Automaton Model. In line with our running example, consider that we need to process a stream of `Temp(area, tmp, ts)` and `Hum(area, pct, ts)` events that measure the temperature and relative humidity in a given area at a certain timestamp `ts`. Fig. 1 shows a portion of such a stream, where `T` abbreviates `Temp` and `H` abbreviates `Hum`. A crucial assumption made by virtually all CER automata models is that the timestamps increase monotonically with the arrival order of events, allowing the automata to treat time simply as an extra event attribute during processing. When events can arrive out of timestamp-order, some form of buffering and re-ordering outside of the automaton model is required.

To illustrate the features that are typically present in CER automaton models, suppose that we want to detect all pairs (t, h) of temperature and humidity events above 40 degrees and below 25%, respectively, that report on the same area ($t.\texttt{area} = h.\texttt{area}$), with h being produced at most 5 minutes after t. This complex event pattern is naturally captured by the non-deterministic automaton shown in Fig. 1 which intuitively operates as follows.

When in state 1 and observing any event (be it `Temp` or `Hum`, represented by $*$), the automaton may non-deterministically choose to stay in state 1, taking the self-loop and discarding the observed event. Alternatively, if the event is a `Temp` event t with $t.\texttt{tmp} > 40$, the automaton can traverse the edge between state 1 and 2. In this case, t's area and timestamp are temporarily stored in internal registers x and y, respectively. In addition, the event itself is buffered so that it can be output later, should a matching `Hum` event be recognized. The self-loop on state 2 indicates that, when in state 2 and observing any event whose timestamp is not 5 minutes greater than the the timestamp of the first matched `Temp` event (stored in register y) the automaton may choose to ignore and discard the event. Alternatively, if the observed event is a `Hum` event h occurring at most 5 minutes later than the timestamp stored in register y, in the area stored in register x and with $h.\texttt{pct} \leq 25$, the automaton can traverse the edge from state 2 to state 3. State 3 is a final state. At this point the automaton can conclude that recognition was successful and output the pair (t, h). Finally, when in state 2 and observing an event whose timestamp is more than 5 minutes later than the first `Temp` event, the automaton must traverse the edge from state 2 to state $\perp$. State $\perp$ is a *failure state* of the automaton, which allows the automaton to conclude that no matter what event comes next, this run can never be successfully completed.

Fig. 1 illustrates four different runs of an automaton on a stream, based on different non-deterministic choices being made. Note that each successful run (i.e., each run which reaches a final state) outputs a new (t, h) pair. Run 4 is a non-successful run. It is important to stress that CER automata, such as the one in Fig. 1, are more powerful than the traditional finite state automata studied in formal language theory:

- They operate on events drawn from an infinite universe of possible events, rather than letters from a finite alphabet. Therefore, the conditions that specify when an edge can be traversed are *formulas* (e.g., `Temp t, t.tmp> 40`) rather than simple letters. Automata in CER systems are hence *symbolic automata* [15] in the sense formal language theory.
- They need to be able to compare data in the attributes of the current event with data of previously observed events. This is required in particular to check timing constraints. Therefore, CER automata are equipped with a finite number of *registers* that can be used to store values when edges are traversed and whose content can be inspected in edge formulas. Automata in CER systems are hence *register automata* in the sense of formal language theory.
- They produce output rather than deciding whether a string is matched or not. In Fig. 1 this is indicated by the `output` directive on transitions. CER automata are hence *transducers*.

All of these features combined make CER automata an excellent formalism to represent combinations of frequently used operators expressible in many complex event languages such as: sequences (as illustrated in Fig. 1) but also co-occurrence, iteration (also known as Kleene closure), and the usual boolean connectives (conjunction, union and bounded forms of negation).

Recognition. CER automata are inherently non-deterministic. As illustrated in Fig. 1, different non-deterministic choices such as ignoring or consuming an event might result in different outputs. To process an event stream, however, we must use a deterministic strategy. Therefore, when using CER automata we must maintain at runtime the set of *all* possible candidate runs. Each run is characterized by its current state, the content of the registers, and the (partial) output that will be generated if it reaches a final state. When processing starts, this set contains only one candidate run (in the initial state, with empty registers and output). When a new event arrives, the set of candidate runs is updated to reflect all the possible transitions. Runs that reach a final state generate the corresponding output and are removed from the set of active runs; runs that have reached a failure state are simply removed without generating output.

It is not hard to see that the set of runs may rapidly become very large [17]: polynomial or even exponential in the amount of events in the time window under consideration. Since each candidate run needs to be inspected and updated for each new event, an explicit representation of all candidate runs may therefore be inefficient. For this reason, numerous systems store the set of candidate runs in a compressed form. Whenever a candidate run reaches a final state, this representation is partially decompressed to construct the output. Multiple forms of compression have been investigated in the literature. For example, SASE [14] *factorizes* commonalities between runs that originated from the same ancestor run. SASE+ [17], in contrast, only stores so-called *maximal runs* from which other runs can be efficiently computed. Even when representing candidate runs in compressed form, however, one risks generating (and updating) many candidate runs that are afterwards discarded without generating output. This may be circumvented by delaying recognition using so-called *lazy automata* [11].

3 TREE-BASED MODELS

While automata-based models have seen wide-spread uptake, some CER systems also employ tree-based models. Again, tree-based formalisms are used for both modelling and recognition, i.e., they may describe the complex event patterns to be recognized as well as the applied recognition algorithm.

Tree-based Event Patterns. In essence, tree-based models for the specification of complex event patterns define a tree of event operators. These operators connect primitive or complex events to form new complex events. A realisation of such a model to pattern specification has been presented in ZStream [13]. Here, the set of supported event operators includes sequencing, negation, conjunction, disjunction, and Kleene closure. Operators may further be assigned constraints (times windows, value predicates), which shall be satisfied when matching events to the children of an operator.

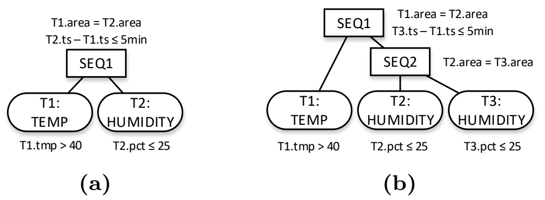

Figure 2: Tree-based models for our running example.

Taking up the above example, the complex event pattern defined in Fig. 1 can also be represented as a tree-based model, see Fig. 2a. That is, the leaf nodes of this tree denote temperature and humidity events above 40 degrees and below 25%, respectively. They are combined by the root of the tree, a sequence operator requiring that the high temperature event is followed by a low humidity event within a window of 5 minutes.

We further note that the general idea of a hierarchical nesting of event operators as put forward by tree-based models for event pattern specification resembles notions of dependency graphs for event patterns. In the E-Cube model [12], for example, such a dependency graph models refinement hierarchies between event patterns, thereby enabling sharing of intermediate results.

Tree-based Recognition. Tree-based models play an important role in recognition algorithms for complex event patterns. They may be used in combination with automata-based models to realise a hybrid processing strategy. Then, automata are used for basic sequence recognition and sets of candidate runs are further pruned using algebraic operators. The initial recognition algorithm of SASE [16] is one example of this, as it first performs the detection of candidate sequences of events before filtering them based on correlation predicates, negation operators, or a time window.

However, it has been argued that such a hybrid approach still suffers from the major drawbacks of an automata-based model, see [13]. That is, automata-based models assume that the order of events in the definition of a complex event pattern (*pattern order*) is also followed in the recognition procedure (*recognition order*). Yet, a recognition order that deviates from the pattern order may enable more efficient event processing, especially if there are large differences in the frequencies of matching events.

Furthermore, the aforementioned approach to model negation in automata-based models by means of failure states, as outlined in Fig. 1, is limited. Transitions to failure states may only be conditioned over data of already observed events, but cannot refer to data of future events. As an example, consider a simple sequence pattern of the form $(A, \neg B, C)$ with a correlation predicate $B.x = C.x$. For this pattern, the construction of a failure state is not possible. Therefore, event recognition needs to rely on a hybrid strategy that filters the candidate sequences of events once they have been detected using an automata-based model.

Against this background, systems such as ZStream [13] and Esper [1] ground their recognition algorithms in trees of event operators. Then, all nodes of this tree are assigned buffers. For leaf nodes, these buffers store the input events as they arrive, whereas the buffers of non-leaf nodes store intermediate results that are assembled from sub-tree buffers. The nesting of operators in the tree then determines the recognition order. For example, the tree of an extended version of our pattern, see Fig. 2b, represents a recognition order, in which sequences of two low humidity events are constructed *before* identifying matching low temperature events.

According to this model, recognition happens by (i) loading a batch of events into the buffers assigned to leaf nodes, immediately evaluating any potential condition over their associated data values; (ii) computing time constraints based on events that are candidates for the last event of a pattern match; (iii) propagating and verifying these constraints at all buffers at leaf nodes; (iv) assembling match results in bottom-up manner based on the semantics of the event operators in the tree.

It is worth to note that the idea of decoupling of pattern order and recognition order as provided by tree-based models can also be incorporated directly in automata-based models. Specifically, tree-structured automata explicitly enumerate all possible recognition orders [11], whereas only one of them, as determined by a cost model, is used for recognition at a any point in time.

4 LOGIC-BASED MODELS

Logic-based CER systems are characterized by a formal semantics expressed in some form of logic, in contrast to other types of CER systems that often present an informal or procedural semantics [5]. In some cases, logic-based CER systems encode rules using logic programming and use inference to detect complex events. For instance, ETALIS [3] builds on the Prolog language and inference mechanism. In other cases, CER rules are converted to other forms to make event recognition more efficient. For instance, T-Rex [7] translates rules into automata.

In the remainder of this section, we first present some logic-based approaches to model CER and then we discuss possible detection methods.

Logic-based Modeling. Several types of logic-based formalisms have been adopted to model CER. Here, we focus on two prominent approaches that we believe are good representatives of logic-based modeling: chronicle recognition [10] and event calculus [4].

Chronicle recognition. Chronicle recognition relies on temporal logic and encodes event occurrences using logic predicates that define the time of occurrence and the content of each event. Complex events are defined starting from primitive ones linked together with contextual and temporal constraints.

An example of CER language that builds on a chronicle recognition formalism is TESLA [7]. For instance, the following TESLA Rule R defines a complex event `WoodFire` when there is a humidity lower than 25% in a certain area and the last measurement of temperature in the same area is higher than 40 degrees.

```
Rule R
define   WoodFire(area: string, temp: double)
from     Humidity(percentage < 25 and area = $a) and
         last Temp(value > 40 and area = $a)
         within 5 min from Humidity
where    area = Temp.area, temp = Temp.value
```

The rule predicates on the occurrence of two events —`Humidity` and `Temp`—, on their content —the `percentage` of `Humidity` must be lower than 25, the `value` of `Temp` must be greater than 40, and they must share the same value of `area`—, and on temporal relations —`Temp` must occur in a window of 5 minutes before `Humidity`. The rule further indicates that a `WoodFire` event contains two attributes and binds their values to the attributes of the `Temp` event that triggered the occurrence.

TESLA translates the constraints expressed in the rules into first-order metric temporal logic formulas that indicate the conditions necessary for the detection of a given composite event. In addition to the types of constraints presented in the previous Rule R, TESLA also offers abstractions to predicate on the absence of events —*negations*— and on aggregate data.

The logic foundation enables a formal specification of the semantics of processing, including selection and consumption policies. Selection refers to the set of events to consider when many of them satisfy the rule constraints. For instance, in the previous Rule R, if multiple `Temp` events with a value greater then 40 are followed by a `Humidity` event, the rule triggers a single occurrence of `WoodFire`, using the *last* occurred `Temp` event. Other possibilities include selecting the *first* occurrent `Temp` event, or even *all* the available `Temp` events to trigger a different `WoodFire` for each of them. Consumption refers to the possibility to re-use the same primitive events for further triggering of the same rule or other rule. TESLA offers an explicit `consuming` clause to indicate all the events that are invalidated after a detection and cannot take part in other detections.

Other prominent examples of languages that build on chronicle recognition logic-based formalisms are Amit [2] and ETALIS [3]. Differently from TESLA, these systems adopt an interval semantics where events have a duration.

Event calculus. Event calculus builds on *fluents*, which are properties that hold different values at different points in time. In event calculus, an *event description* includes rules that define the event occurrences, the effects of events, and the values of fluents [5].

For instance, a CER that builds on event calculus can define `WoodFire` starting from two fluents that encode the current value of temperature and humidity in a given area. The value of these fluents is updated with the arrival of events. `WoodFire` holds when the the value of the humidity fluent is lower than 25% and at the same time the value of the temperature fluent is greater than 40 degrees.

Notably, approaches based on the event calculus formalism can update the recognized events when new information arrives with a delay or old information is revised by the input sources [5].

Logic-based Recognition. Several approaches have been proposed for event recognition in logic-based systems. In the domain of chronicle recognition, a straightforward approach is the use of logic inference mechanisms. For instance, ETALIS translates CER rules in Prolog and uses the Prolog engine for recognition [3]. Other systems translate rules into more efficient structures to perform incremental recognition as new events become available: examples include temporal constraint networks [5] and automata [7]. Besides improving efficiency, incremental recognition enables the CER system to output partially recognized events.

Non-incremental approaches accumulate events and defer their processing as much as possible to optimize memory usage and allow for parallelization [8].

Recognition in event calculus is often performed at query time: events are logged and reasoning is performed on the log when a query is submitted. To overcome this limitation, Artikis et al. propose a *windowing* mechanism that limits the scope of the inference to improve efficiency and scalability [4]. In this context, re-computation from scratch has been proved more efficient than incremental computation, due to the high computational cost of retracting some findings when they are no longer valid.

ACKNOWLEDGEMENTS

A. Artikis is funded by the H2020 project datACRON (687591). M. Ugarte and S. Vansummeren are supported by the Brussels Captial Region–Innoviris (project SPICES). M. Weidlich is supported by the German Research Foundation (DFG), grant WE 4891/1-1.

REFERENCES

[1] Esper. http://esper.codehaus
[2] A. Adi and O. Etzion. 2004. Amit - the situation manager. *The VLDB Journal* 13 (2004), 177–203. Issue 2. http://dx.doi.org/10.1007/s00778-003-0108-y
[3] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer. 2011. *ETALIS: Rule-Based Reasoning in Event Processing.* 99–124.
[4] A. Artikis, M. J. Sergot, and G. Paliouras. 2015. An Event Calculus for Event Recognition. *IEEE Trans. Knowl. Data Eng.* 27, 4 (2015), 895–908.
[5] A. Artikis, A. Skarlatidis, F. Portet, and G. Paliouras. 2012. Logic-Based Event Recognition. *Knowledge Engineering Review* 27, 4 (2012), 469–506.
[6] L. Brenna, A. J. Demers, J. Gehrke, M. Hong, J. Ossher, B. Panda, M. Riedewald, M. Thatte, and W. M. White. 2007. Cayuga: a high-performance event processing engine. In *SIGMOD*. 1100–1102.
[7] G. Cugola and A. Margara. 2010. TESLA: a formally defined event specification language. In *DEBS*. 50–61.
[8] G. Cugola and A. Margara. 2012. Low latency complex event processing on parallel hardware. *JPDC* 72, 2 (2012), 205–218.
[9] G. Cugola and A. Margara. 2012. Processing Flows of Information: From Data Stream to Complex Event Processing. *Comput. Surveys* 44, 3 (2012), 15.
[10] C. Dousson and P. Le Maigat. 2007. Chronicle Recognition Improvement Using Temporal Focusing and Hierarchisation. In *IJCAI*. 324–329.
[11] I. Kolchinsky, I. Sharfman, and A. Schuster. 2015. Lazy evaluation methods for detecting complex events. In *DEBS*. 34–45.
[12] M. Liu, E. A. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta. 2011. E-Cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In SIGMOD, ACM, 889–900.
[13] Y. Mei and S. Madden. 2009. ZStream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD*.
[14] B. Mozafari, K. Zeng, L. D'Antoni, and C. Zaniolo. 2013. High-performance complex event processing over hierarchical data. *ACM TODS* 38, 4 (2013), 21:1–21:39.
[15] G. Van Noord and D. Gerdemann. 2001. Finite state transducers with predicates and identities. *Grammars* 4, 3 (2001), 263–286.
[16] E. Wu, Y. Diao, and S. Rizvi. 2006. High-performance complex event processing over streams. In *SIGMOD*, ACM, 407–418.
[17] H. Zhang, Y. Diao, and N. Immerman. 2014. On complexity and optimization of expensive queries in complex event processing. In *SIGMOD*. 217–228.