PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

A FRAMEWORK FOR COMPLEX EVENT PROCESSING

ALEJANDRO GREZ ARRAU

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Advisor:
CRISTIAN RIVEROS

Santiago de Chile, November 2017

PONTIFICIA UNIVERSIDAD CATOLICA DE CHILE
SCHOOL OF ENGINEERING

A FRAMEWORK FOR COMPLEX EVENT PROCESSING

ALEJANDRO GREZ ARRAU

Members of the Committee:
CRISTIAN RIVEROS
JUAN REUTTER
PABLO BARCELÓ
CHRISTIAN OBERLI

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, November 2017

Gratefully to my family

ACKNOWLEDGEMENTS

I would like to thank, in no particular order:

Cristian Riveros, who has taught me everything I know about this area, abstract and concrete in equal amounts, that is research. Thanks for having the patience to repeat things a thousand times when my ignorance didn't allow me to keep up with what you were saying.

Martín Ugarte, for welcoming me in Brussels and for always being there to guide me, was it for work, to try a good beer or encouraging me to visit the cities of the Old World. If it wasn't for you, I would have been a hermit for two months.

Everyone at the DCC. I can't imagine a more enjoyable place to work.

All the teachers I've had through my life that, thanks to their vocation as educators, instilled in me the love and respect for knowledge. In particular, Miss Práxedes, who showed me that studying is not only about the grades, but much more.

Alonso, Bruno, Gonzalo, Enzo, Tomás and all my friends, for reminding me about the good things in life: enjoy the moment, be happy and share it with others. Music, beer, sports, hobbies, TV shows, movies, lunch; everything is better when shared with you.

Rachel, for your unconditional love and for always believing in me, even when I didn't. "Rest for a while and you will figure out a solution" you told me whenever a problem gave me a hard time. And most of the time you were right.

My brother and sister, for being an unconditional support and a good source of fun. A world without laugh is a world without life.

My parents, who provided me with all the tools I needed to get to be where I am now. You gave me an education that every child should be able to receive and, most importantly, you taught me the values of empathy, perseverance and good will.

Thanks to all of you, for making me who I am now.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Complex Event Processing (CEP) has emerged as the unifying field for technologies that require processing and correlating heterogeneous distributed data in real-time. CEP finds applications in diverse domains, which has resulted in a large amount of proposals for processing complex events. However, existing CEP frameworks are based on ad-hoc solutions that do not rely on solid theoretical ground, making them hard to understand, extend or generalize. Moreover, they are usually presented as informal programming interfaces, and using each of them requires learning a completely new set of skills.

In this thesis we embark on the task of giving a rigorous framework to CEP. As a starting point, we propose a formal language for specifying complex events, called CEPL, that contains the common features used in the literature and has a simple and denotational semantics. We also formalize the so-called *selection strategies*, which are the cornerstone of CEP and had only been presented as by-design extensions to existing frameworks. With a well-defined semantics at hand, we study how to efficiently evaluate CEPL for processing complex events. Towards this goal, we provide optimization results based on rewriting formulas by proposing a normal form for dealing with unary filters. Furthermore, we introduce a formal computational model for CEP based on transducers and symbolic automata, called match automata, that captures the regular core of CEPL, i.e. formulas with unary predicates. By using CEPL rewriting techniques and automata-based translations, we show that formulas in the regular core of CEPL can be evaluated efficiently (constant time per event) when the *next* selection strategy is used. By gathering all these results together, we propose a framework for efficiently evaluating CEPL, establishing solid foundations for future CEP systems.

Keywords: complex event processing, CEP systems, CEP framework, selection strategies, streaming, efficient evaluation, well-defined semantics, automata, logic.

RESUMEN

Complex Event Processing (CEP) ha surgido como el campo unificador para las tecnologías que requieren procesar y correlacionar en tiempo real datos heterogeneos y distribuidos. CEP tiene aplicaciones en diversas areas, lo que ha resultado en que haya un gran numero de propuestas para procesar eventos complejos. Sin embargo, los sistemas CEP existentes están basados en soluciones ad-hoc que no se sustentan en bases teóricas sólidas, lo que los hace difíciles de entender, extender y generalizar. Además, son presentados generalmente de manera informal como iterfaces de programación, y el utilizar cada uno de ellos requiere aprender un conjunto completamente nuevo de conocimientos.

En esta tesis buscamos definir un marco riguroso para CEP. Comenzamos proponiendo un lenguaje formal para especificar eventos complejos, llamado CEPL, que contiene los operadores más comunes utilizados en la literatura y el cual tiene semántica simple y denotacional. Además, formalizamos las llamadas *estrategias de selección*, que son la piedra angular de CEP y en los sistemas existentes son presentadas sólo como extensiones en su diseño. Con la semántica ya definida, estudiamos cómo evaluar eficientemente CEPL. Obtenemos resultados de optimización basados en la reescritura de fórmulas, proponiendo una forma normal para manejar filtros unarios. Además, damos un modelo computacional formal para CEP basado en transductores y autómatas simbólicos, llamado match automata, el cual captura el fragmento regular de formulas con predicados unarios. Utilizando técnicas de reescritura y transformando a autómata, mostramos que el fragmento regular de CEPL puede ser evaluado eficientemente (tiempo constante por evento) cuando se utiliza la estrategia de selección *next*. Con estos resultados, proponemos un marco para evaluar eficientemente CEPL, estableciendo bases sólidas para futuros sistemas CEP.

Palabras Claves: procesamiento de eventos complejos, sistemas CEP, marco CEP, estrategias de selección, streaming, evaluación eficiente, semántica bien definida, autómatas, lógica.

1. INTRODUCTION

The problem of automatically processing continuously arriving information has been present in the database community since the conception of the first Database Management Systems. The so-called Active Database Systems (ADBMS) (Paton & Díaz, 1999) presented a first attempt to solve this problem by allowing users to write *triggers* that are executed upon arrival of tuples. The main goal of ADBMSs was to provide integrity and persistence, focusing on secondary storage (see, e.g., (McCarthy & Dayal, 1989; Gatziu, Fritschi, & Vaduva, 1996)). Naturally, this made ADBMSs poor in terms of performance. Data Stream Management Systems (DSMS) were introduced to work on main memory and overcome this limitation (Golab & Özsu, 2003). Like traditional database management systems, DSMSs are concerned with executing relational queries but over dynamic data (see for example (Chen, DeWitt, Tian, & Wang, 2000; Abadi et al., 2003; Arasu et al., 2003)), and maintaining a *live* version of the results over time. Since DSMSs focus on relational queries over streams, they offer limited reactive capabilities and only see streams as data arriving by parts, and not as a sequence of events (Cugola & Margara, 2012b).

Modern applications must rapidly react to data arriving in high-throughput environments. Moreover, in scenarios like Network Intrusion Detection (Mukherjee, Heberlein, & Levitt, 1994), Industrial Control Systems (Groover, 2007) or Real-Time Analytics (Sahay & Ranjan, 2008), streams must be seen as data events, giving high importance to the order in which the information arrives. Since ADBMSs and DSMSs only fulfill these requirements partially, different communities have proposed domain-specific frameworks and tools for dealing with their particular needs.

Complex Event Processing (CEP) has emerged as the unifying field of technologies for the aforementioned scenarios. From a general perspective, the main requirement of a CEP framework is detecting situations of interest under high-throughput streams. Prominent examples of CEP systems include Sase (Wu, Diao, & Rizvi, 2006), Cayuga (Demers, Gehrke, Hong, Riedewald, & White, 2006), Amit (Adi & Etzion, 2004) and CEDR (Barga, Goldstein, Ali, & Hong, 2007), among others (see (Cugola & Margara, 2012b) for a good

survey). With the objective of making CEP systems applicable to real-life situations, issues like scalability, fault tolerance and distribution have been the main focus of these systems. Other design decisions, like query languages, are generally adapted to match computational models that can efficiently process data (see for example (Zhang, Diao, & Immerman, 2014)). This has produced new data management and optimization techniques, generating promising results in the area (Demers et al., 2006; Barga et al., 2007; Cugola & Margara, 2012a).

Unfortunately, most of CEP systems present solutions for a particular domain. It is hard to find a common theoretical ground, which makes CEP frameworks difficult to understand, extend or generalize. For this reason, they are commonly presented as application programming interfaces, implying that using each of them requires learning a completely new set of skills. Next, we start motivating our work by discussing the current state of CEP systems.

As it has been claimed several times (Galton & Augusto, 2002; Zimmer & Unland, 1999; Cugola & Margara, 2010) the languages for detecting complex events over streams generally lack well-defined denotational semantics. The semantics of several languages are defined either by examples (Luckham, 1996; Adi & Etzion, 2004; Cugola & Margara, 2009), or by intermediate automata models (Wu et al., 2006; Schultz-Møller, Migliavacca, & Pietzuch, 2009; Pietzuch, Shand, & Bacon, 2003). Although there are frameworks that introduce formal semantics (e.g. (Demers et al., 2006; Barga et al., 2007; Akdere, Çetintemel, & Tatbul, 2008; Cugola & Margara, 2010; Anicic et al., 2010)), they do not meet the expectations to pave the foundations of CEP languages. For instance, some of them are too complicated (e.g. sequencing is combined with filters), have unintuitive behavior (e.g. sequencing operator is non-associative), or are severely restricted (e.g. basic operations are supported). As an example, iteration is a fundamental operator in CEP and has not yet been defined successfully as a compositional operator. Since iteration is difficult to define and evaluate, it is usually restricted by not allowing nesting or reuse of variables (Wu et al., 2006; Demers et al., 2006). Thus, without a formal and natural semantics the languages for CEP are in general cumbersome.

The lack of simple denotational semantics also makes it hard to introduce general query optimization techniques. It is common to find complicated heuristics and optimizations that cannot be replicated in other frameworks (see e.g. (Zhang et al., 2014)). Furthermore, optimizations are usually proposed at the architecture level (Mansouri-Samani & Sloman, 1997; Demers et al., 2006; Pietzuch et al., 2003), preventing a unifying optimization theory. This also makes hard to leverage well-known techniques like query rewriting, which is well known in database management systems (Abiteboul, Hull, & Vianu, 1995; Ramakrishnan & Gehrke, 2003). An exception here is (Schultz-Møller et al., 2009) which uses very limited techniques of query rewriting.

Another limitation of existing CEP frameworks is that, for optimization and query evaluation, they used ad-hoc automata models (Demers et al., 2006; Barga et al., 2007; Akdere et al., 2008) without considering previous work in automata theory (Sakarovitch, 2009). These models are usually complicated (Pietzuch et al., 2003; Schultz-Møller et al., 2009), non-standard (Cugola & Margara, 2010; Agrawal, Diao, Gyllstrom, & Immerman, 2008) or informally defined (Demers et al., 2006). For instance, some CEP frameworks enhanced non-deterministic finite state automata with predicates (Agrawal et al., 2008; Schultz-Møller et al., 2009; Pietzuch et al., 2003), buffers (Agrawal et al., 2008), functions (Schultz-Møller et al., 2009), time intervals (Pietzuch et al., 2003), etc. Although some of these features have been studied before in automata theory (Sakarovitch, 2009; Veanes, 2013; Alur & Dill, 1994), they are defined without considering previous work in this field. A proof of this claim is that, although finite state automata is a natural model for CEP, there is no common model for CEP in the literature.

Given this scenario, the main goal of this thesis is to give solid foundations to CEP systems in terms of the query language and query evaluation. Towards these goals, our contributions can be divided in two parts. The first part is dedicated to provide a formal language that allows for expressing the most common features of CEP systems, namely sequencing, filtering, disjunction, and iteration. Inspired in previous CEP frameworks, we introduce CEPL, a logic that contains the main operators found in the literature and has well-defined compositional and denotational semantics. We also formalize the notion

of *selection strategies* which is usually discussed directly (Zhang et al., 2014) or indirectly (Barga et al., 2007) in the literature but has not been properly formalized.

In the second part, we embark on the design of a formal framework for CEPL evaluation. This framework must consider three main building blocks for the efficient evaluation of CEPL: (1) syntactical techniques for rewriting CEPL queries, (2) a well-defined intermediate evaluation model, and (3) efficient translation and algorithms to evaluate this model. About rewriting techniques, we study the structure of CEPL by introducing natural syntactic restrictions (well-formed and safe formulas) and show that these restrictions are relevant for query evaluation. Further, we give a general result on rewriting CEPL formulas into the so-called LP-normal form, a normal form for dealing with unary filters. About the intermediate evaluation model, we introduce a formal computational model for the regular fragment of CEPL, called *match automata*. We show that this model has good properties (e.g. complementation and determinization) and study the evaluation of match automata by showing that a relevant class of this model can be evaluated efficiently in a streaming fashion. We provide algorithms for translating CEPL to match automata. Interestingly, we show that under the *next* selection strategy, formulas in the regular fragment of CEPL can be evaluated efficiently (constant time per event under data complexity) by providing a translation to unambiguous match automata.

Finally, we bring together our results to present a formal framework for evaluating CEPL, and show the main issues for efficient evaluation in CEP systems. Moreover, this framework gives foundations to CEP and settles the bases for future CEP systems.

Organisation. We give an extensive and intuitive introduction to Complex Event Processing and our framework in Section 2. The logic and selection strategies are formalized in Section 3 and 4, respectively. The syntactical structure of the logic is studied in Section 5 and the computational model and its properties are given in Section 6. Section 7 is devoted to the evaluation of the logic with match automata. Section 8 puts all the results in perspective and presents our evaluation framework for CEP Systems. We finally give some concluding remarks in Section 9.

2. EVENTS IN ACTION

In this section we motivate and present the main features and challenges of CEP. The examples used in this section will also serve throughout the thesis as running examples.

In a usual CEP setting, events arrive in a streaming fashion to a system that must detect certain *patterns* (Cugola & Margara, 2012b). For the purpose of illustration assume there is a stream produced by wireless sensors positioned in a farm, whose objective is both to detect fires and achieve optimal irrigation. For the sake of simplification, assume that there are three sensors, and each of them can measure both temperature (in Celsius degrees) and relative humidity (as percentage of vapor in the air). Each sensor is assigned an id in $\{0, 1, 2\}$. The *events* produced by the sensors consist of the id of the sensor and a measurement of temperature or humidity. For the sake of brevity, we write $T(id, tmp)$ for an event reporting temperature tmp from sensor with id id, and similarly $H(id, hum)$ for events reporting humidity. We present such a stream in Figure 2.1, where each column represents an event and the *value* row is the temperature or relative humidity if the event is of type temperature (T) or humidity (H), respectively.

As previously mentioned, complex events are generally specified by domain experts in the form of *patterns*. For the sake of illustration, assume that the position of sensor 0 is particularly prone to fires, and it has been detected that a temperature measurement above 40 degrees Celsius followed by a humidity measurement of less than 25% represents a fire with high probability. Then, such sequence of two events is a complex event of interest. Let us intuitively explain the syntax and semantics with which a domain expert could express this as a pattern (from now on a *formula*) in our framework:

$$\varphi_1 = (T \texttt{ AS } x; H \texttt{ AS } y) \texttt{ FILTER } (x.tmp > 40 \ \wedge y.hum <= 25 \ \wedge x.id = 0 \ \wedge y.id = 0)$$

This formula is asking for two events, one of type temperature (T) and one of type humidity (H). The events of type temperature and humidity are given names x and y, respectively, and the two events are filtered to select only those pairs (x, y) representing a high temperature and low humidity measured by sensor 0. Before defining the semantics

type	H	T	H	H	T	T	T	H	H	...
id	2	0	0	1	1	0	1	1	0	
value	35	45	20	25	40	42	25	70	18	...
index	0	1	2	3	4	5	6	7	8	...

FIGURE 2.1. A stream S of events measuring temperature (T) and humidity (H). "value" contains degrees and humidity for T- and H- events, respectively.

of φ_1, let us discuss what would be the expected result of evaluating this formula over a stream. A first important remark is that event streams are noisy in practice, and one does not expect the events matching a formula to be *contiguous* in the stream. Then, a CEP engine needs to be able to *dismiss* irrelevant events (as opposed to regular expressions). The semantics of the *sequencing* operator (;) will thus allow for arbitrary events to occur in between the events of interest. A second remark is that in CEP the events matching a formula are particularly relevant to the end user. Therefore, every time that a formula *matches* a complex event in the stream, the final user should obtain enough information to retrieve the events that compose the complex event. Therefore, the *output* of evaluating a formula over a stream is a set of *matches*, where each match is the set of indexes (stream positions) of the events that witness the complex event.

We proceed to intuitively explain the evaluation of φ_1 over the stream S (Figure 2.1). Let $S[i]$ be the event with index i in the stream. What we expect as output is a set of pairs $\{i, j\}$ such that $S[i]$ is of type T, $S[j]$ is of type H, $i < j$, and they satisfy the conditions expressed in the FILTER. By inspecting this stream, we can see that the pairs satisfying these conditions are $\{1, 2\}$, $\{1, 8\}$, and $\{5, 8\}$. These are the elements that the user should get as output in order to retrieve the events from the stream for further analysis.

Formula φ_1 illustrates in a simple way the two most elemental features of CEP, namely *sequencing* and *filtering* (Cugola & Margara, 2012b; Arasu et al., 2003; Zhang et al., 2014; Abadi et al., 2003; Buchmann & Koldehofe, 2009). But although it detects a set of possible fires, it restricts the *order* in which the two events must occur, namely the temperature must be measured before the humidity. Naturally, this could prevent the detection of a fire in which the humidity was measured first. This motivates the introduction of *disjunction*, another common feature in CEP engines (Cugola & Margara, 2012b; Arasu et al., 2003).

In our framework, disjunction is expressed by means of the OR operator. To illustrate, we extend φ_1 by allowing events to appear in arbitrary order.

$$\varphi_2 = [(T \texttt{ AS } x; H \texttt{ AS } y) \texttt{ OR } (H \texttt{ AS } y; T \texttt{ AS } x)]$$
$$\texttt{FILTER } (x.tmp > 40 \ \wedge y.hum <= 25 \wedge x.id = 0 \ \wedge y.id = 0)$$

Intuitively, the OR operator allows for any of the two patterns to be matched, and then applies the filter as in φ_1. The result of evaluating φ_2 over the stream S of Figure 2.1 is the same as evaluating φ_1 over S plus the match $\{2, 5\}$.

So far we have illustrated the use of CEP as a mean to raise alerts when a certain complex event occurs, but from a wider scope the objective of CEP is to retrieve information of interest from streams. For example, assume that we want to see how does temperature change in the location of sensor 1 whenever there is a sudden increase of humidity. A problem here is that we don't know a priori the amount of temperature measurements, and therefore we need to capture an unbounded amount of events. An operator for *iteration* (Cugola & Margara, 2012b; Arasu et al., 2003; Wu et al., 2006), commonly denoted by +, is generally introduced in CEP frameworks for solving this problem. The + operator introduces several difficulties in the semantics of CEP languages. For example, since events are not required to occur contiguously in a stream, the nesting of + is particularly tricky and most frameworks simply disallow this (see for example (Wu et al., 2006; Arasu, Babu, & Widom, 2006; Demers et al., 2006)). Coming back to our example, the formula for measuring temperatures whenever a sudden increase of humidity is detected by sensor 1 is:

$$\varphi_3 = [H \texttt{ AS } x; (T \texttt{ AS } y \texttt{ FILTER } y.id = 1)+; H \texttt{ AS } z]$$
$$\texttt{FILTER } (x.hum < 30 \ \wedge z.hum > 60 \wedge x.id = z.id = 1)$$

Intuitively, variables x and z witness a sudden increase of humidity from less than 30% to more than 60%, and y captures temperature measures between x and z. Note that the filter for y is included inside the + operator. Some frameworks allow to declare variables

inside a + and filter them outside that operator (see, e.g., (Wu et al., 2006)). Although it is possible to define the semantics for that syntax (simply as a universal quantifier over the occurrences of the variable), this form of filtering makes the definition of nesting + difficult. Another semantic subtlety of the + operator is the association of y to an event. Given that we want to match the event $(T \texttt{ AS } y \texttt{ FILTER } y.id = 1)$ an unbounded number of times: do we want to associate y to one event or to different events across repetitions? Certainly, we want the latter option since each of the matched temperatures (i.e. T events) will be different. In Section 3, we introduce a natural semantics that allows for nesting arbitrarily many + and associate variables (inside + operators) to different events across repetitions.

Now let us explain the semantics of φ_3 over stream S (Figure 2.1). First, notice that the only two humidity events satisfying the top-most filter are $S[3]$ and $S[7]$. The temperature measurements between these two events are $S[4]$ and $S[6]$. As expected, the match $\{3, 4, 6, 7\}$ is part of the output. However, there are also other matches in the output. Since, as discussed, there might be irrelevant events between relevant ones, the semantics of + must allow for *skipping* arbitrary events. Actually, in the presented match we are already skipping some humidity and temperature events. This implies that, in order to provide well-defined compositional semantics, one must allow for *skipping* events that might be of interest (in our example, temperature measurements of sensor 1). Therefore, the matches $\{3, 6, 7\}$ and $\{3, 4, 7\}$ are also part of the output.

The set of matches generated by formulas φ_1 and φ_3 raises an interesting question: are users interested in receiving all matches? Are some matches more informative than others? Coming back to the output matches of φ_3 ($\{3, 6, 7\}$, $\{3, 4, 7\}$ and $\{3, 4, 6, 7\}$), one can easily argue that the biggest match is more informative than others since all matches are contained in it. A more complicated analysis deserves the matches output by φ_1. In this scenario, the pairs that have the same second component (e.g., $\{1, 8\}$ and $\{5, 8\}$) represent a fire occurring at the same place and time, so one could argue that only one of the two is necessary. Given that $\{1, 8\}$ happens *before* $\{5, 8\}$, a user would probably want $\{1, 8\}$ as the only output match of φ_1 when the last event of the stream is received.

The decision of generating only a subset of the matches, and which subset to return, is generally called a *selection strategy* (Wu et al., 2006; Zhang et al., 2014). A common design across CEP-systems is that formulas are defined to extract all matches and it is responsibility of the users to apply selection strategies over formulas to restrict the set of output matches. Selection strategies are a fundamental feature of CEP but, unfortunately, there is no previous proposal that has defined them formally. A special mention deserves the so-called *next* selection strategy (Wu et al., 2006; Zhang et al., 2014) which in CEP systems usually models the idea of outputting the "most consecutive" match. Although the semantics of next has been proposed or mentioned in previous papers (e.g (Barga et al., 2007)), it is usually defined incorrectly (Wu et al., 2006; Zhang et al., 2014) or across the language making simple operators complicated (Demers et al., 2006). In Section 4 we formally define selection strategies, including *next*. Furthermore, we show in Section 7 that the next selection even allows to optimize the evaluation of formulas.

As it can be deduced from the examples above, the evaluation of CEP formulas can easily become computationally intensive. For example, the + operator allows for a power-set construction, potentially introducing an exponential blowup in the number of results. Therefore, the effectiveness of a CEP framework is based not only on the expressive power of its formulas, but also on the efficiency with which formulas can be evaluated. Because of their similarities with regular expressions, it is common to find automata-based models for evaluating CEP formulas in the literature (Demers et al., 2006; Barga et al., 2007; Akdere et al., 2008). In Section 6, we introduce a model named *match automata* that is based on synchronized transducers (Frougny & Sakarovitch, 1993) and symbolic automata (Veanes, 2013). We also provide a translation from CEP formulas like the ones presented above to match automata.

When evaluating a match automaton over a stream, an important optimization is to stop the *runs* that will not lead to a match as soon as possible. To illustrate this fact, consider again formula φ_1. Syntactically, this formula states "find an event x followed by an event y, and then check that they satisfy the filter conditions". However, we would like an execution engine to only consider those events x with $id = 0$ and whose temperature

measurement is more than 40 degrees. Only afterwards the possible matching events y should be considered. In Section 5 we present rewriting techniques for CEP formulas that allow for this type of optimization. In particular, we present a procedure for *pushing* filter conditions as close to the definition of the variables as possible. For example, formula φ_1 can be restated as

$$\varphi_1' = [(T \texttt{ AS } x) \texttt{ FILTER } (x.tmp > 40 \wedge x.id = 0)]; \\ [(H \texttt{ AS } y) \texttt{ FILTER } (y.hum <= 25 \ \wedge y.id = 0)]$$

In this case the translation is straightforward because the FILTER condition of φ_1 only contains conjunctions. However, when adding logical disjunction the rewriting needs a more involved analysis of the formula.

We conclude this section by illustrating one more common feature of CEP, namely *correlation*. Correlation is introduced by filtering events with predicates that involve more than one event. For example, consider that we want to see how does temperature change at some location whenever there is a sudden increase of humidity there. Then, what we need is a pattern similar to φ_3 where all the events must be produced by the same sensor, but that sensor is not necessarily sensor 1. This is achieved by the following pattern:

$$\varphi_4 = [H \texttt{ AS } x; (T \texttt{ AS } y \texttt{ FILTER } y.id = x.id)+; H \texttt{ AS } z] \\ \texttt{FILTER } (x.hum < 30 \ \wedge z.hum > 60 \wedge x.id = z.id)$$

Notice that here the filters contain the binary predicates $x.id = y.id$ and $x.id = z.id$ that force all events to have the same id. Although this might seem simple, the evaluation of formulas that correlate events introduces new challenges. Intuitively, formula φ_4 is more complicated in the sense that the value of x must be remembered and used during the evaluation in order to compare it with all the incoming events. If the reader is familiar with automata theory (Hopcroft & Ullman, 1979; Sakarovitch, 2009), this behavior is clearly not "regular" and it will not be captured by a finite state model. In this thesis, we want to study and characterize the regular part of CEP-systems. Therefore, in Sections 6

and 7 we restrict our analysis to formulas with unary predicates (like φ_1, φ_2 and φ_3) that capture the regular core of CEP-languages, and postpone the analysis of formulas like φ_4 for future work. It is important to mention here that the semantics of our language proposal (plus the selection strategies and rewriting of formulas) is defined in general and not restricted to any subfragment.

We have illustrated sequencing, filtering, disjunction, iteration and correlation, and we discussed optimization techniques and features of CEP that will be further developed in the rest of the thesis. In the next section we proceed to formally define the syntax and semantics of CEP formulas.

3. A QUERY LANGUAGE FOR CEP

In this section we present a formal language for specifying complex events called *CEP-logic*. We first introduce the basic notions and then proceed to define the operations found regularly in the literature (Section 3.2). Finally, in Section 3.3 we introduce operators that provide further CEP features but are found less frequently in the literature.

3.1. Basic Definitions

Let $\mathbf{A}$ be a set of *attribute names* and let $\mathbf{D}$ be a set of values. A relation name R is any finite subset of $\mathbf{A}$. If R is a relation name, then an R-tuple is a function $t : R \to \mathbf{D}$. We say that the type of an R-tuple t is R, and denote this by $\mathrm{type}(t) = R$. A database schema $\mathcal{R}$ (or just schema) is a finite set of relation names. For any relation name R, $\mathrm{tuples}(R)$ denotes the set of all possible R-tuples, i.e., $\mathrm{tuples}(R) = \{t : R \to \mathbf{D}\}$. Similarly, for any database schema $\mathcal{R}$, $\mathrm{tuples}(\mathcal{R})$ is the set of all R-tuples for $R \in \mathcal{R}$.

Given a schema $\mathcal{R}$, an *$\mathcal{R}$-stream* S is an infinite sequence $S = t_0 t_1 \ldots$ where $t_i \in \mathrm{tuples}(\mathcal{R})$. When $\mathcal{R}$ is clear from the context, we refer to S simply as a stream. Given a stream $S = t_0 t_1 \ldots$ and a position $i \in \mathbb{N}$, the i-th element of S is denoted by $S[i] = t_i$, and the sub-stream $t_i t_{i+1} \ldots$ of S is denoted by S_i. Here, we suppose that the order of the sequence implicitly defines an order among tuples and we usually call $S[i]$ an *event* of S at time i. Furthermore, contrary to other frameworks (Pietzuch et al., 2003) we consider that the time of each event is implicitly given by the order of the stream and we do not consider extensions like intervals. We leave these extensions for future work (see Section 9).

Let $\mathbf{X}$ be a set of variables and $\mathbf{P}(\mathcal{R})$ a set of predicates over $\mathrm{tuples}(\mathcal{R})$, where each $P \in \mathbf{P}(\mathcal{R})$ has arity $\mathrm{arity}(P)$. For the sake of simplification, for each $P \in \mathbf{P}(\mathcal{R})$ we write $P(x_1, \ldots, x_n)$, where $n = \mathrm{arity}(P)$ and $x_1, \ldots, x_n \in \mathbf{X}$. We define the set $\mathbf{F}(\mathcal{R})$ of *selection formulas* over $\mathcal{R}$ as the smallest set of formulas such that $\mathbf{P}(\mathcal{R}) \subseteq \mathbf{F}(\mathcal{R})$ and is closed under conjunction, disjunction and negation. For example, if $\mathbf{P}(\mathcal{R})$ contains the predicates $P_1(x) := x.hum < 30$, $P_2(z) := z.hum > 60$ and $P_3(x, z) := x.id = z.id$, then the outer-most filter of φ_4 (see Section 2) is a formula in $\mathbf{F}(\mathcal{R})$.

An *assignment* is a partial function $\sigma : \mathbf{X} \rightharpoonup \text{tuples}(\mathcal{R})$. Given an assignment σ and a predicate $P(x_1, \ldots, x_n)$ in $\mathbf{P}(\mathcal{R})$, we say that σ satisfies P (denoted by $\sigma \models P$) if $P(\sigma(x_1), \ldots, \sigma(x_n))$ evaluates to true. For every formula in $\mathbf{F}(\mathcal{R})$ that is not a predicate, the semantics is defined recursively as usual. Finally, for the computational complexity analysis we assume that given an assignment σ and $\alpha \in \mathbf{F}(\mathcal{R})$, it takes time $O(1)$ to verify whether $\sigma \models \alpha$.

3.2. Core CEP Logic

Now we proceed to give the syntax of what we call the *core* of CEP-logic (core-CEPL for short), a logic inspired by previous CEP frameworks (e.g. (Wu et al., 2006; Demers et al., 2006; Barga et al., 2007)). This language features those operations commonly found in the literature.

The set of formulas in core-CEPL, or core formulas for short, is given by the following grammar:

$$\varphi \;:=\; R \,\texttt{AS}\, x \;\mid\; \varphi \,\texttt{FILTER}\, \alpha \;\mid\; \varphi \,\texttt{OR}\, \varphi \;\mid\; \varphi\,;\,\varphi \;\mid\; \varphi\texttt{+}$$

where R is a relation name, x is a variable in $\mathbf{X}$ and α is a selection formula in $\mathbf{F}(\mathcal{R})$. As opposed to existing frameworks, we do not restrict the use of variables, or nesting of operators. In particular, we allow for arbitrary nesting of +.

Now we proceed to define the semantics of core formulas, for which we need to introduce some further notation. A match M is defined as a non-empty and finite set of natural numbers. As mentioned in the previous section, a match contains the positions that witness the satisfaction of a formula over a stream, and moreover, they are the final output of evaluating a formula over a stream. We denote by $|M|$ the size of a match M and by $\min(M)$ and $\max(M)$ the minimum and maximum element of M, respectively. Given a stream $S = t_0 t_1 \ldots$ and $M = \{i_1, i_2, \ldots, i_n\}$ with $i_j < i_{j+1}$, the subsequence $t_{i_1} t_{i_2} \ldots t_{i_n}$ of S is denoted by $S[M]$. Intuitively, if $S[i]$ is an event of S, $S[M]$ represents a *complex event*. Given two matches M_1 and M_2, we denote by $M_1 \cdot M_2$ the *concatenation* of two matches, that is, $M_1 \cdot M_2 := M_1 \cup M_2$ whenever $\max(M_1) < \min(M_2)$ and empty otherwise.

In core-CEPL formulas, variables are second class citizens because they are only used to filter and select particular events in a stream, i.e. they are not retrieved as part of the output. As examples in Section 2 suggest, we are only concerned with finding the positions that witness the match and not which position corresponds to which variable. The reason behind this is that the operator + allows for repetitions, and therefore variables under a (possibly nested) +-operator would need to have a special meaning, particularly for filtering operations. This discussion motivates the following definitions. Given a core-CEPL formula φ we denote by $\mathrm{var}(\varphi)$ the set of all variables that appear in φ (i.e. as R `AS` x or in a selection formula α) and by $\mathrm{vdef}(\varphi)$ all variables defined in φ by a clause of the form R `AS` x. Furthermore, we denote by $\mathrm{vdef}_+(\varphi)$ all variables in $\mathrm{vdef}(\varphi)$ that are defined outside the scope of all +-operators. For example, in the formula $\varphi = (T \texttt{ AS } x\,;\, (H \texttt{ AS } y)+) \texttt{ FILTER } z.id = 1$ we have that $\mathrm{var}(\varphi) = \{x, y, z\}$, $\mathrm{vdef}(\varphi) = \{x, y\}$, and $\mathrm{vdef}_+(\varphi) = \{x\}$.

The last notion needed for defining the semantics of core-CEPL is how to assign variables to events. Here, the notion of assignments introduced in Section 3.1 is not enough for the semantics since they assign tuples to variables loosing the relative position of data inside a stream. In other words, two tuples in a stream can be equal with respect to its content (i.e. data) but they will be different with respect to its position. For this reason, we want to assign positions to variables instead of just tuples. Formally, a valuation is a partial function $\nu : \mathbf{X} \rightharpoonup \mathbb{N}$. Given a stream S, a valuation ν naturally induces an assignment ν_S from variables to $\mathrm{tuples}(\mathcal{R})$ defined by $\nu_S(x) = S[\nu(x)]$ for every $x \in \mathrm{dom}(\nu)$. Finally, given a finite subset $U \subseteq \mathbf{X}$ and two valuations ν_1 and ν_2, we define the valuation $\nu_1[\nu_2 \to U]$ by $\nu_1[\nu_2 \to U](x) = \nu_2(x)$ if $x \in U$ and $\nu_1[\nu_2 \to U](x) = \nu_1(x)$ otherwise.

Now we are ready to define the semantics of a core-CEPL formula φ. Given a match M, a stream S, and a position $i \in \mathbb{N}$, we say that M belongs to the evaluation of φ over S starting at position i and under the valuation ν (denoted by $M \in [\![\varphi]\!](S, i, \nu)$) if one of the following conditions holds:

- $\varphi = R$ `AS` x, $M = \{\nu(x)\}$, $\mathrm{type}(S[\nu(x)]) = R$ and $i \leq \nu(x)$.

- $\varphi = \rho$ `FILTER` α, $M \in [\![\rho]\!](S, i, \nu)$ and $\nu_S \vDash \alpha$.
- $\varphi = \rho_1$ `OR` ρ_2 and ($M \in [\![\rho_1]\!](S, i, \nu)$ or $M \in [\![\rho_2]\!](S, i, \nu)$).
- $\varphi = \rho_1 \,;\, \rho_2$ and $M = M_1 \cdot M_2$ for two matches M_1 and M_2 such that $M_1 \in [\![\rho_1]\!](S, i, \nu)$ and $M_2 \in [\![\rho_2]\!](S, j, \nu)$, with $j = \max(M_1) + 1$.
- $\varphi = \rho+$ and there is a valuation ν' such that either $M \in [\![\rho]\!](S, i, \nu[\nu' \to U])$ or $M \in [\![\rho \,;\, \rho+]\!](S, i, \nu[\nu' \to U])$, where $U = \mathrm{vdef}_+(\rho)$.

There are a couple of important remarks here. First, notice that the valuation ν can be defined over a superset of the variables mentioned in the formula. This is important for the sequencing operator (;) because we require the matches from both sides to be produced with the same valuation. Second, when we evaluate a subformula of the form $\rho+$, we *carry* the value of variables defined outside the subformula. For example, the subformula $(T$ `AS` y `FILTER` $y.id = x.id)+$ of φ_4 does not define the variable x. However, from the definition of the semantics we see that x will be *already assigned* (because R `AS` x occurs in the upper level). This is precisely where other frameworks fail to formalize iteration, as without this construct it is not easy to correlate the variables inside + with the ones outside, as we illustrate in φ_4.

Notice also that the sequencing operator (;) is associative. Although this might seem natural, there are CEP frameworks with formal semantics in which this is not the case (see, e.g., (Demers et al., 2006)). This is one of the reasons to include the position i in our definition, as it restricts the matches produced by the right-hand side of a sequence only to those occurring after the left-hand side was matched. Also, this will allow us to give compositional semantics to *selectors* (Section 4).

As it was previously mentioned, in a core-CEPL formula variables are just used for comparing attributes with `FILTER` and are not relevant for the final output. To this end, we say that M belongs to the evaluation of φ over S, denoted by $M \in [\![\varphi]\!](S)$, if there exists a valuation ν such that $M \in [\![\varphi]\!](S, 0, \nu)$, namely, we evaluate φ over S starting from position 0. As an example, the reader can check that the matches presented in Section 2 are indeed matches of φ_1 to φ_3 over the stream of sensors measurements.

3.3. Other operators

We now extend the syntax and semantics of core-CEPL by adding new operators. Some of these operators are natural extensions of the core language and others have been proposed in previous work (Barga et al., 2007; Adi & Etzion, 2004). Specifically, the syntax of the extended core CEP-logic (or ecore-CEPL) is given by extending the grammar of core-CEPL with the following operators:

$$\varphi := \varphi \texttt{ AND } \varphi \mid \varphi \texttt{ ALL } \varphi \mid \varphi \texttt{ UNLESS } \varphi$$

We call φ an ecore-CEPL formula (or simply ecore formula).

Similar to core-CEPL, we define the semantics of ecore-CEPL over a stream S by using the notions of matches and valuations. The semantics of the core operators are as defined in Section 3.2, and the semantics of a formula $\varphi = \rho_1 \texttt{ OP } \rho_2$, where OP is any of the new operators AND, ALL, and UNLESS, is defined recursively as follows: given a match M, a stream S, a position i and a valuation ν, we say that $M \in [\![\varphi]\!](S, i, \nu)$ if one of the following conditions holds:

- OP = AND, $M \in [\![\rho_1]\!](S, i, \nu)$ and $M \in [\![\rho_2]\!](S, i, \nu)$.
- OP = ALL and $M = M_1 \cup M_2$ for two M_1 and M_2 such that both $M_1 \in [\![\rho_1]\!](S, i, \nu)$ and $M_2 \in [\![\rho_2]\!](S, i, \nu)$ hold.
- OP = UNLESS, $M \in [\![\rho_1]\!](S, i, \nu)$ and for all M' and ν' with $\min(M) \leq \min(M') \leq \max(M') \leq \max(M)$, it holds that $M' \notin [\![\rho_2]\!](S, i, \nu[\nu' \to \mathrm{vdef}_+(\rho_2)])$.

The AND operator simply selects those matches produced by both formulas. Although this is natural for sets, it is very restrictive for matching events. On the contrary, ALL is more flexible and allows to combine two matches. In this sense, ALL is similar to sequencing but allows that the matches occur at any point in time, even overlapping and intersecting. For example, formula φ_2 of Section 2 asks for a temperature measurement and a humidity measurement that can occur in any order and satisfy a certain condition. This formula could have been written more succinctly as $[(T \texttt{ AS } x) \texttt{ ALL } (H \texttt{ AS } y)] \texttt{ FILTER } (\ldots)$. The

objective of the UNLESS is to introduce negation. It is important to mention that the *negated* formula (the right-hand side) is restricted to matches between the start and end of matches for the formula in the left-hand side. This is motivated by the fact that a match should not depend on objects that are distant in the stream. For example, consider that we want to see a drastic increase in temperature. This can be expressed as a sequence of a low temperature (less than 20 degrees) and a high temperature (more than 40 degrees), where no other temperatures occur in between. This can be expressed by the following pattern:

$$\big[(T \texttt{ AS } x) \texttt{ FILTER } (x.tmp < 20); (T \texttt{ AS } y) \texttt{ FILTER } (y.tmp > 40)\big]$$
$$\texttt{UNLESS}\big[(T \texttt{ AS } z) \texttt{ FILTER } (z.tmp >= 20 \wedge z.tmp <= 40)\big]$$

We stress that valuations are not part of the output, and as in core-CEPL we write $M \in [\![\varphi]\!](S)$ when there is a valuation ν such that $M \in [\![\varphi]\!](S, 0, \nu)$.

4. SELECTION STRATEGIES

Matching complex events is usually a computationally intensive task. As our running example and the definition of ecore-CEPL might suggest, the main reason behind this is that the amount of matches can grow exponentially in the size of the stream, forcing systems to process large numbers of *candidate* matches. In order to optimize the matching processes, it is common to restrict the set of results (Carlson & Lisper, 2010; Wu et al., 2006; Zhang et al., 2014). This is one of the cornerstones of CEP systems, and most of the proposals in the literature introduce them simply as ad-hoc extensions matching particular computational models. For a more general approach, we introduce the so-called *selection strategies* as unary operators (called *selectors*). Formally, we define the syntax of full CEP-logic, or simply CEPL, by the following grammar:

$$\begin{aligned} \varphi \;\; := \;\; & R \,\texttt{AS}\, x \mid \varphi \,\texttt{FILTER}\, \alpha \mid \varphi \,\texttt{OR}\, \varphi \mid \varphi \,;\, \varphi \mid \varphi + \\ & \varphi \,\texttt{AND}\, \varphi \mid \varphi \,\texttt{ALL}\, \varphi \mid \varphi \,\texttt{UNLESS}\, \varphi \\ & \texttt{STRICT}(\varphi) \mid \texttt{NXT}(\varphi) \mid \texttt{MAX}(\varphi) \end{aligned}$$

We now proceed to define the semantics of the selectors, starting with the strict-contiguity selector `STRICT`. Recall that formula φ_1 in Section 2 detects complex events composed by a temperature above 40 degrees Celsius followed by a humidity of less than 25%. As already argued, in general one could expect many other events between x and y. However, it could be the case that this particular pattern is of interest only if the events occur contiguously in the stream, namely a temperature right after a humidity measure. The strict-contiguity selector `STRICT` only allows strictly consecutive matches. Formally, for any CEPL formula φ we have that $M \in [\![\texttt{STRICT}(\varphi)]\!](S, i, \nu)$ holds if $M \in [\![\varphi]\!](S, i, \nu)$ and for every $i, j \in M$, there is no $k \in \mathbb{N} \setminus M$ such that $i < k < j$ (i.e., M is an interval). For example, in our running example $\texttt{STRICT}(\varphi_1)$ would only produce the match $\{1, 2\}$, although $\{1, 8\}$ and $\{5, 8\}$ are also matches for φ_1 over S.

The `STRICT` contiguity selector is generally included in CEP frameworks because it allows to carry over good properties from regular expressions. However, for reasons we have already discussed it is not particularly interesting to capture contiguous events in streams. In CEP one would like to have more flexible selection strategies that allow for obtaining fewer yet meaningful results. One notion that appears often in the literature is that of selecting only those matches that are *as consecutive as possible*. For example, consider again the pattern φ_1 and the stream S from Section 2. As we discussed, both $\{1, 8\}$ and $\{5, 8\}$ are matches for φ_1 over S. Now, if a user didn't want to obtain all matches, which of these two matches would he prefer? Here is where the notion of "most consecutive" appears. It is widely accepted that the first match is preferred over the second, since the first event of $\{1, 8\}$ occurred before the first event of $\{5, 8\}$, and therefore the first match is *more consecutive* in the stream. Another intuition behind this notion is that it is better to match the *next* event instead of skipping it and selecting another event in the future.

The above example motivates the semantics of the *next* selection strategy, which has been discussed and used in most of the CEP systems (Cugola & Margara, 2012b) as an special operator (Barga et al., 2007; Wu et al., 2006; Zhang et al., 2014) or behind the semantics of the sequencing operator (Demers et al., 2006). However, they fail to define it correctly since they either mix the semantics of selectors with the semantics of sequencing or iteration, or they define the selector semantics for a restricted set of operators (e.g. they cannot support Kleene closure). In our framework, we formalize the semantics of the next operator based on a special order over matches. This allows us to capture the intuition of "more consecutive" reflected in the literature, while giving a general definition.

Let M_1 and M_2 be two matches. The symmetric difference between M_1 and M_2 is denoted by $M_1 \vartriangle M_2$ and is the set of all elements either in M_1 or M_2 but not in both. We say that $M_1 \leq_{\text{next}} M_2$ if either $M_1 = M_2$ or $\min(M_1 \vartriangle M_2) \in M_2$. For example, we have that $\{5, 8\} \leq_{\text{next}} \{1, 8\}$ since the minimum element in $\{5, 8\} \vartriangle \{1, 8\} = \{1, 5\}$ is 1, which is in $\{1, 8\}$. Notice that the $\subseteq$-relation is a refinement of the $\leq_{\text{next}}$-relation in the sense that if $M_1 \subseteq M_2$ then $M_1 \leq_{\text{next}} M_2$. This follows the intuition that the more elements a match

has, the *more consecutive* it is. Moreover, one can prove that the $\leq_{\text{next}}$-relation forms a total order among matches, implying the existence of a maximum over any finite set of matches.

Lemma 4.1. *$\leq_{\text{next}}$ is a total order between matches.*

PROOF. For $\leq_{\text{next}}$ to be a total order between matches, it has to be *reflexive* (trivial), *anti-symmetric*, *transitive*, and *total*. The proof for each property is given next.

Anti-symmetric. Consider any two matches M_1 and M_2 such that $M_1 \leq_{\text{next}} M_2$ and $M_2 \leq_{\text{next}} M_1$. $M_2 \leq_{\text{next}} M_1$ means that either $M_1 = M_2$ or (1) $\min\{(M_1 \cup M_2) - (M_1 \cap M_2)\} \in M_1$, and $M_1 \leq_{\text{next}} M_2$ that either $M_2 = M_1$ or (2) $\min\{(M_2 \cup M_1) - (M_2 \cap M_1)\} \in M_2$. If (1) were true, it would mean that (2) could not be true, so $M_2 = M_1$ would have to be true, becoming a contradiction. So, the only possible scenario is that $M_1 = M_2$.

Transitivity. Consider any three matches M_1, M_2 and M_3 such that $M_1 \leq_{\text{next}} M_2$ and $M_2 \leq_{\text{next}} M_3$. $M_1 \leq_{\text{next}} M_2$ means that either $M_1 = M_2$ or (1) $\min\{(M_1 \cup M_2) - (M_1 \cap M_2)\} \in M_2$. If $M_1 = M_2$, then $M_1 \leq_{\text{next}} M_3$ because $M_2 \leq_{\text{next}} M_3$. Now, if $M_1 \neq M_2$, then (1) must hold, which means that the lowest element that is either in M_1 or M_2, but not in both, has to be in M_2. Let's call this element l_1. $M_2 \leq_{\text{next}} M_3$ means that either $M_2 = M_3$ or (2) $\min\{(M_2 \cup M_3) - (M_2 \cap M_3)\} \in M_3$. Again, if $M_2 = M_3$, then $M_1 \leq_{\text{next}} M_3$ because $M_1 \leq_{\text{next}} M_2$. Now, if $M_2 \neq M_3$, then (2) must hold, which means that the lowest element that is either in M_2 or M_3, but not in both, has to be in M_3. Let's call this element l_2.

Given that $M_1 \neq M_2$ and $M_2 \neq M_3$, define for every $i \in \{1, 2, 3\}$ and $j \in \{1, 2\}$ the set $M_i^{<l_j}$ as the set of elements of M_i which are lower than l_j, i.e., $M_i^{<l_j} = \{x \mid x \in M_i \wedge x < l_j\}$. It is clear that $M_1^{<l_1} = M_2^{<l_1}$ and $M_2^{<l_2} = M_3^{<l_2}$, because of (1) and (2), respectively. Also, because of (2) it holds that $l_2 \notin M_2$, so $l_1 \neq l_2$.

Consider first the case where $l_1 < l_2$. This means that (3) $M_1^{<l_1} = M_3^{<l_1}$. Moreover, if l_1 were not in M_3, it would contradict (2), so (4) $l_1 \in M_3$ must hold. With (3) and (4), it follows that l_1 is the lowest element that is either in M_1 or M_3 but not in both, and it is in M_3. This proves that $\min\{(M_1 \cup M_3) - (M_1 \cap M_3)\} \in M_3$, and thus $M_1 \leq_{\text{next}} M_3$.

Now consider the case where $l_2 < l_1$. This means that (5) $M_1^{<l_2} = M_3^{<l_2}$. Because l_2 is not in M_2, it cannot be in M_1, otherwise it would contradict (1), so (6) $l_2 \notin M_1$ must hold. Also, because of (2) we know that (7) $l_2 \in M_3$ must hold. With (5), (6) and (7), it follows that l_2 is the lowest element that is either in M_1 or M_3 but not in both, and it is in M_3. This proves that $\min\{(M_1 \cup M_3) - (M_1 \cap M_3)\} \in M_3$, and thus $M_1 \leq_{\text{next}} M_3$.

Total. Consider any two matches M_1 and M_2. If $M_1 = M_2$, then $M_1 \leq_{\text{next}} M_2$ holds. Consider now the case where $M_1 \neq M_2$. Define the set $M = (M_1 \cup M_2) \backslash (M_1 \cap M_2)$ which is the set of all elements either in M_1 or M_2, but not in both. Because $M_1 \leq_{\text{next}} M_2$, there must be at least one element in M. In particular, this implies that there is a minimum element l in M. If l is in M_2, then $M_1 \leq_{\text{next}} M_2$ holds, and if l is in M_1, then $M_2 \leq_{\text{next}} M_1$ holds. □

We now define the semantics of the next selector $\texttt{NXT}(\varphi)$: for any CEPL formula φ we have $M \in [\![\texttt{NXT}(\varphi)]\!](S, i, \nu)$ if $M \in [\![\varphi]\!](S, i, \nu)$ and for every match M' such that $M <_{\text{next}} M'$ and $\max(M) = \max(M')$, it holds that $M' \notin [\![\varphi]\!](S, i, \nu)$. In our running example, $\{1, 8\}$ satisfies $\texttt{NXT}(\varphi_1)$ on S, as there is no "more consecutive" match satisfying φ_1 in the same prefix. Note that we compare matches with respect to $\leq_{\text{next}}$ that have the same final position. This ensures that the maximum match always exists and that the optimality of a match only depends on the matches over the same prefix.

Another selector that has been proposed in the literature is that of selecting only the maximal matches in terms of inclusion. This corresponds to obtaining those matches that are *as informative as possible*, and therefore contain the biggest sets of events. Formally, for any CEPL formula φ we have that $M \in [\![\texttt{MAX}(\varphi)]\!](S, i, \nu)$ holds iff $M \in [\![\varphi]\!](S, i, \nu)$ and for all matches M' such that $M \subset M'$ and $\max(M) = \max(M')$, it holds that $M' \notin [\![\varphi]\!](S, i, \nu)$. Coming back to our example, the MAX selector will output both matches $\{1, 8\}$ and $\{5, 8\}$ for φ_1, given that both matches are maximal in terms of set inclusion. On the contrary, formula φ_3 produced the matches $\{3, 6, 7\}$, $\{3, 4, 7\}$, and $\{3, 4, 6, 7\}$. Then if we evaluate $\texttt{MAX}(\varphi_3)$ over the same stream, we will obtain only $\{3, 4, 6, 7\}$ as output, which is the maximal match. It is interesting to note that if we evaluate $\texttt{NXT}(\varphi_3)$ over the

stream we will also get $\{3, 4, 6, 7\}$ as the only output, illustrating that next yields matches with maximal information.

So far we have extensively discussed the foundations of CEP. We presented a formal language with well-defined semantics that contains most of the operators found in the literature, including the so-called selection strategies. This is an important and foundational first step, but is not enough for defining a complete and practical framework for CEP. In the rest of the thesis we study several practical aspects of CEPL. We start by discussing the syntactic form of CEPL formulas, and define syntactic restrictions that characterize semantic properties of interest. Then, we present a computational model and show how CEPL formulas in the introduced syntactic fragments can be evaluated in this model. Moreover, we identify a fragment of CEPL that can be efficiently evaluated. Finally, we put all pieces together and present a complete framework for evaluating the studied CEPL formulas in practice.

5. SYNTACTIC ANALYSIS OF CEPL

In this section we study the syntactic form of CEPL formulas, and define the classes of *well-formed* formulas and *safe* formulas. These classes are based on syntactic restrictions that characterize semantic properties of interest. Then, we define a convenient normal form for CEPL and show that any formula can be rewritten in this form.

5.1. Syntactic restrictions of formulas

The definition of CEPL provides well-defined semantics for all formulas, allowing for a more concise theoretical analysis. However, there are some formulas whose semantics can be unintuitive. Consider for example the formula:

$$\varphi_5 = (H \texttt{ AS } x) \texttt{ FILTER } (y.tmp \leq 30).$$

Here, x will be naturally bounded to the only element in a match, but y will not *add* a new position to a match. By the semantics of CEPL, a valuation ν for φ_5 must assign a position for y that satisfies the filter, but such position is not restricted to occur in the match. Moreover, y is not necessarily bounded to any of the events seen up to the last element in the match, and thus a match could depend on future events. For example, if we evaluate φ_5 over our running example S (Figure 2.1), we have that $\{2\} \in [\![\varphi_5]\!](S)$, however, a streaming evaluation of φ_5 would have to wait until the event at position 6 to output this match.

To avoid formulas with unintuitive semantics we define a natural notion of *well-formed* formulas. As the previous example illustrates, this requires defining where variables are *bounded* by a sub-formula of the form $R \texttt{ AS } x$. The set of bound variables of a formula φ is denoted by $\mathrm{bound}(\varphi)$ and is recursively defined as follows:

- $\mathrm{bound}(R \texttt{ AS } x) = \{x\}$
- $\mathrm{bound}(\rho \texttt{ FILTER } \alpha) = \mathrm{bound}(\rho)$
- $\mathrm{bound}(\rho_1 \texttt{ OR } \rho_2) = \mathrm{bound}(\rho_1) \cap \mathrm{bound}(\rho_2)$

- $\text{bound}(\rho+) = \varnothing$
- $\text{bound}(\rho_1 \texttt{ UNLESS } \rho_2) = \text{bound}(\rho_1)$
- $\text{bound}(\rho_1 \texttt{ OP } \rho_2) = \text{bound}(\varphi_1) \cup \text{bound}(\varphi_2)$
- $\text{bound}(\texttt{SEL}(\rho)) = \text{bound}(\rho)$

where $\texttt{OP} \in \{\texttt{;}, \texttt{,}, \texttt{AND}, \texttt{ALL}\}$ and SEL is any selection strategy. Note that for the OR operator a variable must be defined in both formulas in order to be bounded. Similarly, in the case of UNLESS the variables that count are the ones in φ_1 since φ_2 is just checking that some matches do not exist. We say that CEPL formula φ is *well-formed* if for every sub-formula of the form ρ FILTER α and every variable $x \in \text{var}(\alpha)$, there is another sub-formula ρ_x such that $x \in \text{bound}(\rho_x)$ and ρ is a sub-formula of ρ_x. Note that this definition allows for including filters with variables defined in a wider scope. For example, formula φ_4 in Section 2 is well-formed although variable x is used in the filter $y.id = x.id$ and defined outside the +-operator.

As it was previously discussed, we would like to consider only CEPL formulas that can output matches as soon as the last position of the match is seen. We formalize this with the notion of *streamable* formulas. Given streams S_1 and S_2, we say that S_1 is equal to S_2 up to position i, denoted by $S_1 =_i S_2$, if $S_1[j] = S_2[j]$ for each $j \leq i$. Then, we say that a formula φ is *streamable* if for every match M and stream $S = t_0 t_1 \ldots$ it holds that $M \in [\![\varphi]\!](S)$ if, and only if, $M \in [\![\varphi]\!](S')$ for every stream S' such that $S =_{\max(M)} S'$. In other words, streamable formulas can (in principle) be evaluated in a streaming fashion given that the belonging of M to $[\![\varphi]\!](S)$ only depends on the prefix $t_0 t_1 \ldots t_{\max(M)}$.

The next result shows that if we restrict to the class of well-formed formulas, we do have the streamable property.

Theorem 5.1. *Every well-formed formula is streamable.*

PROOF. Let φ be a well-formed formula. In order to prove that φ is streamable we first define the following lemmas:

Lemma 5.1. *Consider any CEPL formula φ, stream S, match M, valuation ν, and $i \in \mathbb{N}$. If $M \in [\![\varphi]\!](S, i, \nu)$, then $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.*

PROOF. We prove this by induction over the formula φ:

- Consider $\varphi = R$ `AS` x. Then, $\text{bound}(\varphi) = x$ and, by definition, $M \in [\![\varphi]\!](S, i, \nu)$ implies that $M = \{\nu(x)\}$. Therefore, the lemma holds.
- Consider $\varphi = \rho$ `FILTER` α. Then, $M \in [\![\varphi]\!](S, i, \nu)$ implies that $M \in [\![\rho]\!](S, i, \nu)$, therefore by induction hypothesis $\nu(x) \in M$ for every $x \in \text{bound}(\rho)$. Moreover $\text{bound}(\varphi) = \text{bound}(\rho)$, thus $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.
- Consider $\varphi = \rho_1$ `OR` ρ_2. Then, $M \in [\![\varphi]\!](S, i, \nu)$ implies that either $M \in [\![\rho_1]\!](S, i, \nu)$ or $M \in [\![\rho_2]\!](S, i, \nu)$. Without loss of generality, assume that it is the first case. Then, by induction hypothesis $\nu(x) \in M$ for every $x \in \text{bound}(\rho_1)$. Moreover, because $\text{bound}(\varphi) = \text{bound}(\rho_1 \cap \rho_2)$, then $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.
- Consider $\varphi = \rho_1 \,;\, \rho_2$. Then, $M \in [\![\varphi]\!](S, i, \nu)$ implies that there exist matches M_1 and M_2 with $M = M_1 \cdot M_2$ such that $M_1 \in [\![\rho_1]\!](S, i, \nu)$ and $M_2 \in [\![\rho_2]\!](S, \max(M_1) + 1, \nu)$. By induction hypothesis $\nu(x) \in M_i$ for every $x \in \text{bound}(\rho_i)$. Because $\text{bound}(\varphi) = \text{bound}(\rho_1) \cup \text{bound}(\rho_2)$ and both $M_1, M_2 \in M$, it holds that $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.
- Consider $\varphi = \rho+$. By definition, $\text{bound}(\varphi) = \varnothing$, therefore the lemma trivially holds.
- Consider $\varphi = \rho_1$ `AND` ρ_2. Then, by definition $M \in [\![\varphi]\!](S, i, \nu)$ means that both $M \in [\![\rho_1]\!](S, i, \nu)$ and $M \in [\![\rho_2]\!](S, i, \nu)$ hold. Therefore, by induction hypothesis $\nu(x) \in M$ for every $x \in \text{bound}(\rho_1) \cup \text{bound}(\rho_2) = \text{bound}(\varphi)$.
- Consider $\varphi = \rho_1$ `ALL` ρ_2. By definition, $M \in [\![\varphi]\!](S, i, \nu)$ means that there exist matches M_1 and M_2 such that $M = M_1 \cup M_2$ and both $M \in [\![\rho_1]\!](S, i, \nu)$ and $M \in [\![\rho_2]\!](S, i, \nu)$ hold. Then, by induction hypothesis $\nu(x) \in M_i$ for every $x \in \text{bound}(\rho_i)$. Moreover, because $M = M_1 \cup M_2$ and $\text{bound}(\varphi) = \text{bound}(\rho_1) \cup \text{bound}(\rho_2)$, it holds that $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.

- Consider $\varphi = \rho_1$ `UNLESS` ρ_2. By definition, $M \in [\![\varphi]\!](S, i, \nu)$ implies $M \in [\![\rho_1]\!](S, i, \nu)$, therefore by induction hypothesis $\nu(x) \in M$ for every $x \in \text{bound}(\rho_1)$. Moreover, $\text{bound}(\varphi) = \text{bound}(\rho_1)$, therefore $\nu(x) \in M$ for every $x \in \text{bound}(\varphi)$.

Given that the lemma holds in all cases, the lemma is shown. □

For the next lemma, define the set $\text{unbound}(\varphi) \subseteq \mathbf{X}$ such that $x \in \text{unbound}(\varphi)$ if there exist a sub-formula of the form φ' `FILTER` α, $x \in \text{var}(\alpha)$ and there does not exist another sub-formula φ_x such that $x \in \text{bound}(\varphi_x)$ and φ' is a sub-formula of φ_x. In other words, we define the set $\text{unbound}(\varphi)$ of all variables that witness that a formula is not well-formed.

Lemma 5.2. *Consider any CEPL formula φ, stream S, match M, valuation ν, and $i, j \in \mathbb{N}$. If $M \in [\![\varphi]\!](S, i, \nu)$, $m \leq j$ for every $m \in M$ and $\nu(x) \leq j$ for all $x \in \text{unbound}(\varphi)$, then $M \in [\![\varphi]\!](S', i, \nu)$ for every stream S' such that $S =_j S'$.*

PROOF. We prove this by induction over φ:

- Consider $\varphi = R$ `AS` x. Then, by definition $M \in [\![\varphi]\!](S, i, \nu)$ means that $M = \{\nu(x)\}$, $\text{type}(S[\nu(x)]) = R$ and $i \leq \nu(x)$. Moreover, because $\nu(x) \leq j$ then $S'[\nu(x)] = S[\nu(x)]$, thus $\text{type}(S'[\nu(x)]) = R$ and $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho$ `FILTER` α. Then, $M \in [\![\varphi]\!](S, i, \nu)$ implies that $M \in [\![\rho]\!](S, i, \nu)$, and $\text{unbound}(\varphi) = \text{unbound}(\rho)$, thus $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho)$. Consider any stream S' such that $S =_j S'$. Then, by induction hypothesis $M \in [\![\rho]\!](S', i, \nu)$. Because of Lemma 5.1, for every $x \in \text{bound}(\varphi)$ it holds that $\nu(x) \leq j$. Notice that every variable x in α is either bounded or unbounded, therefore it holds that $\nu(x) \leq j$ for every x in α. Moreover, because $\nu_S \vDash \alpha$ then $\nu_{S'} \vDash \alpha$, thus $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho_1$ `OR` ρ_2. Then, $M \in [\![\varphi]\!](S, i, \nu)$ means that either $M \in [\![\rho_1]\!](S, i, \nu)$ or $M \in [\![\rho_2]\!](S, i, \nu)$. Without loss of generality, assume that it is the first case. Because $\text{unbound}(\rho_1) \subseteq \text{unbound}(\varphi)$, $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho_1)$. Consider any stream S' such that $S =_j S'$. Then, by induction hypothesis $M \in [\![\rho_1]\!](S', i, \nu)$, thus $M \in [\![\varphi]\!](S', i, \nu)$.

- Consider $\varphi = \rho_1 \,;\, \rho_2$. Then, $M \in [\![\varphi]\!](S, i, \nu)$ implies that there exist matches M_1 and M_2 with $M = M_1 \cdot M_2$ such that $M_1 \in [\![\rho_1]\!](S, i, \nu)$ and $M_2 \in [\![\rho_2]\!](S, \max(M_1) + 1, \nu)$. Moreover, because $\nu(x) \leq j$ for all $x \in \text{unbound}(\varphi)$ then $\nu(x_1) \leq j$ for all $x_1 \in \text{unbound}(\rho_1) \setminus \text{bound}(\rho_2)$. Also, if $x \in \text{unbound}(\rho_1) \cap \text{bound}(\rho_2)$, then by Lemma 5.1 it holds that $x \in M_2$. Moreover, $m \leq j$ for every $m \in M$ and $M_2 \subseteq M$, therefore $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho_1)$, and similarly $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho_2)$. Consider any stream S' such that $S =_j S'$. Then, by induction hypothesis $M_1 \in [\![\rho_1]\!](S', i, \nu)$ and $M_2 \in [\![\rho_2]\!](S', \max(M_1) + 1, \nu)$, thus $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho+$. Then it holds that $\text{unbound}(\rho) = \text{unbound}(\varphi)$, therefore $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho)$. By definition, there exists $\nu' \in \text{val}(M)$ such that either $M \in [\![\rho]\!](S, i, \nu[\nu' \to U])$ or $M \in [\![\rho \,;\, \rho+]\!](S, i, \nu[\nu' \to U])$ where $U = \text{vdef}_+(\rho)$. Consider any stream S' such that $S =_j S'$. By induction hypothesis, the first and second cases directly imply that $M \in [\![\rho]\!](S', i, \nu[\nu' \to U])$ and $M \in [\![\rho \,;\, \rho+]\!](S', i, \nu[\nu' \to U])$ respectively, thus $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho_1 \texttt{ AND } \rho_2$. Then, by definition $M \in [\![\varphi]\!](S, i, \nu)$ means that both $M \in [\![\rho_1]\!](S, i, \nu)$ and $M \in [\![\rho_2]\!](S, i, \nu)$ hold. Similarly to the ; case, by Lemma 5.1 it holds that $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho_1) \cup \text{unbound}(\rho_1)$. Consider any stream S' such that $S =_j S'$. By induction hypothesis $M \in [\![\rho_1]\!](S', i, \nu)$ and $M \in [\![\rho_2]\!](S', i, \nu)$, thus $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho_1 \texttt{ ALL } \rho_2$. By definition, $M \in [\![\varphi]\!](S, i, \nu)$ means that there exist matches M_1 and M_2 such that $M = M_1 \cup M_2$ and both $M \in [\![\rho_1]\!](S, i, \nu)$ and $M \in [\![\rho_2]\!](S, i, \nu)$ hold. Again, similarly to the ; case it holds by Lemma 5.1 that $\nu(x) \leq j$ for all $x \in \text{unbound}(\rho_1) \cup \text{unbound}(\rho_1)$. Consider any stream S' such that $S =_j S'$. Therefore, by induction hypothesis $M_1 \in [\![\rho_1]\!](S', i, \nu)$ and $M_2 \in [\![\rho_2]\!](S', i, \nu)$, thus $M \in [\![\varphi]\!](S', i, \nu)$.
- Consider $\varphi = \rho_1 \texttt{ UNLESS } \rho_2$. By definition, $M \in [\![\varphi]\!](S, i, \nu)$ implies $M \in [\![\rho_1]\!](S, i, \nu)$ and for all M' and ν' such that $\min(M) \leq \min(M')$ and $\max(M') \leq \max(M)$, $M' \notin [\![\rho_2]\!](S, i, \nu[\nu' \to \text{vdef}_+(\rho_2)])$. Moreover, $\text{unbound}(\rho_2) \subseteq \text{unbound}(\varphi) \cup \text{bound}(\rho_1)$ and $\text{unbound}(\rho_1) \subseteq \text{unbound}(\varphi)$, therefore $\nu(x) \leq j$ holds for all $x \in \text{unbound}(\rho_1) \cup$

$\mathrm{unbound}(\rho_2)$. Consider any stream S' such that $S =_j S'$. Then, by induction hypothesis $M \in [\![\rho_1]\!](S', i, \nu)$. Moreover, if there were an M' and ν' such that $\min(M) \leq \min(M')$, $\max(M') \leq \max(M)$ and $M' \in [\![\rho_1]\!](S', i, \nu')$ by induction hypothesis it would mean that $M' \in [\![\rho_1]\!](S, i, \nu')$ which is a contradiction, therefore there exist no such M' and ν'. Then, it holds that $M \in [\![\varphi]\!](S', i, \nu)$.

Given that the lemma holds in all cases, the lemma is shown. □

Now, with Lemma 5.2 the proof is straightforward: because φ is well-formed then $\mathrm{unbound}(\varphi) = \varnothing$, and because of Lemma 5.2 (with $i = 0$ and $j = \max(M)$), for every match M and stream S it holds that $M \in [\![\varphi]\!](S)$ if, and only if, $M \in [\![\varphi]\!](S')$ for every stream S' such that $S =_{\max(M)} S'$, thus φ is streamable. □

One can easily argue that it would be desirable for a CEP-system to restrict the users to only write well-formed formulas. Indeed, the well-formed property can be checked efficiently by a syntactic parser and users should understand that all variables in a formula must be correctly defined. Given that well-formed formulas are streamable and have a well-defined variable structure, in the future we restrict our analysis to well-formed formulas.

Another issue for CEPL is that the reuse of variables can easily produce unsatisfiable formulas. For example, the formula $\psi = T \texttt{ AS } x \,;\, T \texttt{ AS } x$ is not satisfiable (i.e. $[\![\psi]\!](S) = \varnothing$ for every S) because variable x cannot be assigned to two different positions in the stream. This issue arises when variables are reused on conjunctive operators like sequencing (;) or `ALL`. On the other hand, we do not want to be too conservative and disallow the reuse of variables in the whole formula (e.g. φ_2 in Section 2 will not be permitted). This motivates the notion of *safe* CEPL formulas. We say that a CEPL formula is *safe* if for every subformula of the form $\varphi_1 \texttt{ OP } \varphi_2$ with $\texttt{OP} \in \{ \texttt{;}, \texttt{,}, \texttt{AND}, \texttt{ALL} \}$ it holds that $\mathrm{vdef}_+(\varphi_1) \cap \mathrm{vdef}_+(\varphi_2) = \varnothing$. For example, all CEPL formulas introduced so far are safe except for ψ.

The safe notion is a mild but useful restriction to help the evaluation of CEPL and can effectively be checked during parsing time. However, safe formulas are a subset of CEPL

and it could be the case that this prevents users from writing certain patterns. We show in the next result that this is never the case for the core fragment. Formally, we say that two CEPL formulas φ and ψ are equivalent, denoted by $\varphi \equiv \psi$, if for every stream S and match M, it is the case that $M \in [\![\varphi]\!](S)$ if, and only if, $M \in [\![\psi]\!](S)$.

Theorem 5.2. *Let φ be a core-CEPL formula. Then, there is a safe formula φ' such that $\varphi \equiv \varphi'$ and $|\varphi'|$ is at most exponential in $|\varphi|$.*

PROOF. To prove this theorem, we first show that one can push disjunction (by means of OR) to the top-most level of every core-CEPL formula. Formally, we say that a CEPL formula φ is in disjunctive-normal form if $\varphi = (\varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n)$, where for each $i \in \{1, \ldots, n\}$, it is the case that:

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

Now we show that every formula can be translated into disjunctive normal form.

Lemma 5.3. *Every formula φ in core-CEPL can be translated into disjunctive-normal form in time at most exponential $|\varphi|$.*

PROOF. We proceed by induction over the structure of φ.

- If $\varphi = R \texttt{ AS } x$, then φ is already free of OR.
- If $\varphi = \varphi_1 \texttt{ OR } \varphi_2$, the result readily follows from the induction hypothesis.
- If $\varphi = (\varphi')$+, by induction hypothesis φ can be translated into disjunctive normal form.
- If $\varphi = \varphi' \texttt{ FILTER } \alpha$, we know by induction hypothesis that φ' is equivalent to a formula $(\varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n)$. Therefore, φ is equivalent to $(\varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n) \texttt{ FILTER } \alpha$. We show that this latter formula is equivalent to $(\varphi_1 \texttt{ FILTER } \alpha) \texttt{ OR } \cdots \texttt{ OR } (\varphi_n \texttt{ FILTER } \alpha)$. Let S be a stream and assume $M \in [\![(\varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n) \texttt{ FILTER } \alpha]\!](S)$. Then, there is a valuation ν such that $M \in [\![(\varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n)]\!](S, 0, \nu)$ and $\nu_S \vDash \alpha$. By definition of OR, this implies that there is an $i \in \{1, \ldots, n\}$ such that $M \in [\![(\varphi_i)]\!](S, 0, \nu)$.

As $\nu_S \vDash \alpha$, we have $M \in [\![(\varphi_i)\ \texttt{FILTER}\ \alpha]\!](S, 0, \nu)$. We can then immediately conclude that $M \in [\![(\varphi_1\ \texttt{FILTER}\ \alpha)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_n\ \texttt{FILTER}\ \alpha)]\!](S, 0, \nu)$, and therefore $M \in [\![(\varphi_1\ \texttt{FILTER}\ \alpha)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_n\ \texttt{FILTER}\ \alpha)]\!](S)$. The converse follows from an analogous argument.

- If $\varphi = (\varphi_1; \varphi_2)$, by induction hypothesis we know that φ_1 is equivalent to a formula $(\varphi_1^1\ \texttt{OR} \cdots \texttt{OR}\ \varphi_n^1)$ and φ_2 is equivalent to a formula $(\varphi_1^2\ \texttt{OR} \cdots \texttt{OR}\ \varphi_m^2)$. Let φ' be

$$\varphi' = (\varphi_1^1; \varphi_1^2)\ \texttt{OR}\ (\varphi_1^1; \varphi_2^2)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_1^1; \varphi_m^2)\ \texttt{OR}\ (\varphi_2^1; \varphi_1^2)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_2^1; \varphi_m^2)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_n^1; \varphi_1^2)\ \texttt{OR} \cdots \texttt{OR}\ (\varphi_n^1; \varphi_m^2).$$

  We show that $\varphi \equiv \varphi'$. Let S be a stream and let M be a match. If $M \in [\![\varphi]\!](S)$, then there is a valuation ν and two matches M_1 and M_2 such that $M = M_1 \cdot M_2$, $M_1 \in [\![\varphi_1]\!](S, 0, \nu)$ and $M_2 \in [\![\varphi_2]\!](S, 0, \nu)$. Then, there are two numbers i and j such that $M_1 \in [\![\varphi_i^1]\!](S, 0, \nu)$ and $M_2 \in [\![\varphi_j^2]\!](S, 0, \nu)$. As $M = M_1 \cdot M_2$, it immediately follows that $M \in [\![\varphi_i^1; \varphi_j^2]\!](S)$, and thus $M \in [\![\varphi']\!](S)$.

  For the converse assume $M \in [\![\varphi']\!](S)$. Then, there is a valuation ν, a match M and two numbers i and j such that $M \in [\![\varphi_i^1; \varphi_j^2]\!](S, 0, \nu)$. Therefore there are two matches M_1 and M_2 such that $M = M_1 \cdot M_2$, $M_1 \in [\![\varphi_i^1]\!](S, 0, \nu)$ and $M_2 \in [\![\varphi_j^2]\!](S, 0, \nu)$. By semantics of OR, we have $M_1 \in [\![\varphi_1]\!](S, 0, \nu)$ and $M_2 \in [\![\varphi_2]\!](S, 0, \nu)$. As $M = M_1 \cdot M_2$, it readily follows that $M \in [\![\varphi_1; \varphi_2]\!](S) = [\![\varphi]\!](S)$.

□

Having this result, we proceed to show that a core-CEPL formula in disjunctive normal form can be translated into a safe formula. To this end, we need to show the following two lemmas.

Lemma 5.4. *Let φ be a core-CEPL formula in which every* OR *occurs inside the scope of a + operator, and let $x \in \mathrm{vdef}_+(\varphi)$. Then, for every match M, valuation ν, stream S and $i \in \mathbb{N}$ such that $M \in [\![\varphi]\!](S, i, \nu)$, it is the case that $x \in \mathrm{dom}(\nu)$ and $\nu(x) \in M$.*

PROOF. We proceed by induction on the structure of φ. Let ν be a valuation, S a stream, $i \in \mathbb{N}$ and M a match.

- Assume $\varphi = R \texttt{ AS } x$ and that $M \in [\![\varphi]\!](S, i, \nu)$. By definition, we have $M = \{\nu(x)\}$.
- Assume $\varphi = \varphi' \texttt{ FILTER } \alpha$ and that $M \in [\![\varphi]\!](S, i, \nu)$. Let $x \in \mathrm{vdef}_+(\varphi)$. By definition, we have that $M \in [\![\varphi']\!](S, i, \nu)$. Since $x \in \mathrm{vdef}_+(\varphi')$, by induction hypothesis we have $x \in \mathrm{dom}(\mu)$ and $\nu(x) \in M$.
- If $\varphi = (\varphi')+$ the condition trivially holds as $\mathrm{vdef}_+(\varphi') = \varnothing$.
- If $\varphi = \varphi_1; \varphi_2$, then $x \in \mathrm{vdef}_+(\varphi_1)$ or $x \in \mathrm{vdef}_+(\varphi_2)$. Assume w.l.o.g. that $x \in \mathrm{vdef}_+(\varphi_1)$. If $M \in [\![\varphi]\!](S, i, \nu)$, then $M = M_1 \cdot M_2$, where $M_1 \in [\![\varphi_1]\!](S, i, \nu)$. As $x \in \mathrm{vdef}_+(\varphi_1)$, by induction hypothesis we have that $x \in \mathrm{dom}(\nu)$ and $\nu(x) \in M_1 \subseteq M$, concluding the proof.

□

Lemma 5.5. *Let φ be a core-CEPL formula in which every* OR *occurs inside the scope of a + operator, and let S be a stream. If φ has a subformula φ' that is not under the scope of a + operator such that $[\![\varphi']\!](S) = \varnothing$, then $[\![\varphi]\!](S) = \varnothing$.*

PROOF. We proceed by induction on the structure of φ. Let S a stream and assume φ' is a subformula of φ such that $[\![\varphi']\!](S) = \varnothing$. We assume that φ' is a proper subformula, as otherwise the result immediately follows. For this reason, we can trivially skip the case when $\varphi = R \texttt{ AS } x$ or $\varphi = (\varphi_1)+$.

- If $\varphi = \varphi_1; \varphi_2$, then φ' is a subformula of φ_1 or of φ_2. Assume w.l.o.g. that φ' is a subformula of φ_1. By induction hypothesis, as $[\![\varphi']\!](S) = \varnothing$ we have that $[\![\varphi_1]\!](S) = \varnothing$, which immediately implies that $[\![\varphi]\!](S) = \varnothing$.
- If $\varphi = \varphi_1 \texttt{ FILTER } \alpha$, we know that φ' is a subformula of φ_1. By induction hypothesis we have $[\![\varphi']\!](S) = \varnothing$ and by definition of FILTER we obtain $[\![\varphi]\!](S) = \varnothing$.

□

Now we are ready to show that any core-CEPL formula in disjunctive-normal form can be translated into a safe formula, and moreover, this can be done in linear time.

Lemma 5.6. *Let φ be a core-CEPL formula in disjunctive-normal form. Then φ can be translated in linear time into a safe core-CEPL formula φ'.*

PROOF. Assume that $\varphi = \varphi_1 \texttt{ OR } \cdots \texttt{ OR } \varphi_n$ is a core-CEPL formula in disjunctive-normal form. By induction, we assume that every sub-formula of the form $(\varphi')+$ is already safe. Now we show that every unsafe φ_i is unsatisfiable, and therefore it can be safely removed from the disjunction. Proceed by contradiction and assume φ_i is unsafe and satisfiable. Then, it must contain a subformula of the form $\psi_1;\psi_2$ occurring outside the scope of all + operators, and such that $\mathrm{vdef}_+(\psi_1) \cap \mathrm{vdef}_+(\psi_2) \neq \varnothing$. Let $x \in \mathrm{vdef}_+(\psi_1) \cap \mathrm{vdef}_+(\psi_2)$. By Lemma 5.5, we know that $\psi_1;\psi_2$ must be satisfiable. Therefore, there is a stream S, a valuation ν and a mapping M such that $M \in [\![\psi_1;\psi_2]\!](S,0,\nu)$. This implies the existence of two matches M_1 and M_2 such that $M_1 \in [\![\psi_1]\!](S,0,\nu)$ and $M_2 \in [\![\psi_2]\!](S,0,\nu)$. Since $x \in \mathrm{vdef}_+(\psi_1)$ and ψ_1 can only mention OR inside a + operator, by Lemma 5.4 we obtain that $\nu(x) \in M_1$. Similarly, as $x \in \mathrm{vdef}_+(\psi_2)$, we have $\nu(x) \in M_2$. But as $M = M_1 \cdot M_2$, we have that $M_1 \cap M_2 = \varnothing$, contradicting the facts that $\nu(x) \in M_1$ and $\nu(x) \in M_2$.

We have obtained that if any disjunct is unsafe, it cannot produce any results. Therefore, as safeness is easily verifiable, the result readily follows by removing the unsafe disjuncts of φ. Notice that this need to be done in a bottom-up fashion, starting from the subformulas of the form $(\varphi')+$. □

Theorem 5.2 occurs as a corollary of Lemmas 5.3 and 5.6. Indeed, given a core-CEPL formula φ, one can construct in exponential time an equivalent core-CEPL formula φ' in disjunctive normal form. Then, from φ' one can construct in linear time a safe formula in core-CEPL ψ that is equivalent to φ, which is exactly what we wanted to show. □

By this result, for core-CEPL we can restrict our analysis to safe formulas without losing expressiveness of the language. Instead, if we do not impose the safe restriction,

we will have to assume an exponential blow-up in the rewriting of formulas (see Section 8 for further discussion).

5.2. LP-normal form

In this subsection we study how to rewrite CEPL formulas in order to simplify the evaluation of unary filters. Intuitively, filter operators in a CEPL formula can become difficult to handle for a CEP query engine. As it was previously motivated by formula φ_1 and φ_1' in Section 2, it is easier for a query optimizer to evaluate formulas where unary predicates are applied directly over their variables (e.g. φ_1') and not anywhere in the formula (e.g. φ_1). This motivates the definition of formulas in *locally parametrized* normal form (LP-normal form). Let $\mathbf{P}_u(\mathcal{R})$ be the set of all predicates $P \in \mathbf{P}(\mathcal{R})$ such that $\text{arity}(P) = 1$. Furthermore, define $\mathbf{F}_u(\mathcal{R}) \subseteq \mathbf{F}(\mathcal{R})$ to be the set of all selection formulas constructed from atomic predicates in $\mathbf{P}_u(\mathcal{R})$. Then we say that a formula φ is in *LP-normal form* if the following condition holds: for every sub-formula φ' `FILTER` α of φ, if α contains at least one predicate in $\mathbf{P}_u(\mathcal{R})$, then $\varphi' = R$ `AS` x for some R and x, and $\alpha \in \mathbf{F}_u(\mathcal{R})$ with $\text{var}(\alpha) = \{x\}$. In other words, all filters containing unary predicates are applied directly to the definitions of their variables. For instance, formula φ_1' is in LP-normal form while formulas φ_1 and φ_2 are not. Note that non-unary predicates are not restricted, and they can be used anywhere in the formula.

One can easily see the advantage for the query engine of using only formulas in LP-normal form (see Section 7 for further discussion). However, formulas that are not in LP-normal form can still be very useful for declaring patterns. To illustrate this, consider the formula:

$$\varphi_6 = (T \texttt{ AS } x); ((T \texttt{ AS } y \texttt{ FILTER } x.temp \geq 40) \texttt{ OR } (H \texttt{ AS } y \texttt{ FILTER } x.temp < 40))$$

Here, the `FILTER` operator works like a conditional statement: if the x-temperature is greater than 40, then the following event should be a temperature, and a humidity event otherwise. This kind of conditional statements can be very useful for users and a serious problem for query engines. Fortunately, the next result shows that one can always rewrite

a formula to an equivalent LP-normal form formula with an exponential blow-up in the size of the formula.

Theorem 5.3. *Let φ be a CEPL formula. Then, there is a CEPL formula ψ in LP-normal form such that $\varphi \equiv \psi$, and $|\psi|$ is at most exponential in $|\varphi|$.*

PROOF. First we give some definitions to simplify notation. Consider a formula α that has only unary predicates and such that negations are only used over predicates. From now on we consider all formulas to be in this form, since every formula can be written this way by pushing negations inside of the formula and changing $\vee$'s with $\wedge$'s and vice versa. We will also refer to α as the set literals that appear in α, namely, the set of atomic formula or its negation (for consistency, if $p(x)$ appears in α only as $\neg p(x)$, we do not consider $p(x)$). Also, we consider only unary predicates, since these are the ones that we need to modify in order for the formula to be in LP-normal form. We use the notation of α as formula and set indistinctly whenever its meaning is clear from the context. Given a CEPL formula of the form $\varphi = \varphi'$ `FILTER` α, we define the set of *unbounded predicates* of φ , written as $\mathrm{unbound_p}(\varphi)$, as all the predicates (and negations) of the filters that are not instantiated, i.e., $p(x) \in \mathrm{unbound_p}(\varphi)$ if $p(x) \in \alpha$ and $x \notin \mathrm{bound}(\varphi')$. Notice that, as expected, if φ is well-formed then $\mathrm{unbound_p}(\varphi) = \varnothing$, but this does not apply to subformulas, i.e., there could be a subformula φ' of φ such that $\mathrm{unbound_p}(\varphi') \neq \varnothing$.

Consider a well-formed CEPL formula φ with unary predicates. We first provide a construction for a CEPL formula in LP normal form and then prove that it is equivalent to φ. The first step of the construction is focused on rewriting the formula in a way that for every subformula φ' it holds that $\mathrm{unbound_p}(\varphi') = \varnothing$. The construction we provide to achieve this is the following. For every subformula of the form φ' `FILTER` α and every predicate $p(x) \in \mathrm{unbound_p}(\varphi')$, let φ_x be the lowest subformula of φ where x is defined and that has φ' as a subformula. Here we use the fact that φ is well-formed, which means that such φ_x must exist. Then, we rewrite the subformula φ_x inside φ as φ_x^t `FILTER` $p(x)$ `OR` φ_x^f `FILTER` $\neg p(x)$, where φ_x^t and φ_x^f are the same as φ_x but replacing $p(x)$ with `TRUE` and `FALSE`, respectively.

Now that we moved each predicate up to a level where all its variables are defined, the next step is to move each one down to its variable's definition. A first approach is to take every predicate $p(x)$ that appears and move it down to every place where it was defined, i.e., to every subformula of the form R `AS` x. The problem with this is that it would be forcing $p(x)$ to be true, even though this might not be necessary, for example if $p(x)$ appears in one side of a propositional disjunction. To solve, this we first need to "unfold" the filters of the formula, which is done by rewriting each subformula recursively in the following way:

- φ' `FILTER` $\alpha_1 \wedge \alpha_2$ is replaced by $(\varphi'$ `FILTER` $\alpha_1)$ `FILTER` α_2.
- φ' `FILTER` $\alpha_1 \vee \alpha_2$ is replaced by φ' `FILTER` α_1 `OR` φ' `FILTER` α_2.

Notice that, after doing this, all filters have only one predicate, so $p(x)$ can no longer appear inside a propositional disjunction. Now moving down each predicate is done straightforward. For every subformula of the form φ' `FILTER` $p(x)$, the $p(x)$ filter is removed from φ' and instead applied over every subformula of φ' with the form R `AS` x, rewriting it as R `AS` x `FILTER` $p(x)$. Because this step moved every predicate to its definition, the resulting formula is clearly in LP normal form, completing the construction.

Now we prove that the construction above satisfies the lemma, i.e., $[\![\varphi_{lp}]\!](S) = [\![\varphi]\!](S)$ for every stream S, where φ_{lp} is the resulting formula after doing the construction. To prove that the first part does not change the semantics, we show that it stays the same after each iteration. Consider a subformula φ' `FILTER` α and a predicate $p(x) \in \mathrm{unbound_p}(\varphi')$. In particular, the only part modified is φ_x, so it suffices to prove that $M \in [\![\varphi_x]\!](S, i, \nu)$ holds iff $M \in [\![\varphi_x^t$ `FILTER` $p(x)$ `OR` φ_x^f `FILTER` $\neg p(x)]\!](S, i, \nu)$. Let S, i, M, ν be any stream, position, match and valuation, respectively, such that $M \in [\![\varphi_x]\!](S, i, \nu)$. If $\nu_S \vDash p(x)$, then it is enough to prove that $M \in [\![\varphi_x^t]\!](S, i, \nu)$. In a similar way, the only part in which φ_x^t differs with φ_x is that $p(x)$ was set true in α (let α^t be the result of doing this). Therefore, it is enough to prove that, for any j, M' and φ', if $\nu'_S \vDash p(x)$ holds, then $M' \in [\![\varphi'$ `FILTER` $\alpha]\!](S, j, \nu')$ iff $M' \in [\![\varphi'$ `FILTER` $\alpha^t]\!](S, j, \nu')$. This is clearly true since $p(x) \Rightarrow (\alpha \Leftrightarrow \alpha^t)$ is a tautology. Notice that we can assure $\nu'_S \vDash$

$p(x)$ holds because $\nu_S \vDash p(x)$ holds and, when evaluating this part of the formula, the mapping for x must stay the same, otherwise x must have been inside a +-operator, which cannot be the case because $x \in \text{bound}(\varphi_x)$. Moreover, ν' has to be equal to ν. The proof for the case $\nu_S \vDash \neg p(x)$ is similar considering φ_x^f instead of φ_x^t, thus $M \in [\![\varphi_x^t \texttt{ FILTER } p(x) \texttt{ OR } \varphi_x^f \texttt{ FILTER } \neg p(x)]\!](S, i, \nu)$. For the opposite direction, let S, i, M, ν be any stream, position, match and valuation, respectively, such that M belongs to $[\![\varphi_x^t \texttt{ FILTER } p(x) \texttt{ OR } \varphi_x^f \texttt{ FILTER } \neg p(x)]\!](S, i, \nu)$. Then, by definition either $M \in [\![\varphi_x^t \texttt{ FILTER } p(x)]\!](S, i, \nu)$ or $M \in [\![\varphi_x^f \texttt{ FILTER } \neg p(x)]\!](S, i, \nu)$ hold. Without loss of generality, consider the former case, which implies that $\nu_S \vDash p(x)$. By the same reasoning above $M' \in [\![\varphi' \texttt{ FILTER } \alpha]\!](S, j, \nu')$ iff $M' \in [\![\varphi' \texttt{ FILTER } \alpha^t]\!](S, j, \nu')$, hence $M \in [\![\varphi_x]\!](S, i, \nu)$. It is the same for $\nu_S \vDash \neg p(x)$, thus $M \in [\![\varphi_x]\!](S, i, \nu)$ iff M belongs to $[\![\varphi_x^t \texttt{ FILTER } p(x) \texttt{ OR } \varphi_x^f \texttt{ FILTER } \neg p(x)]\!](S, i, \nu)$. Therefore, if we name φ_1 as the result of applying the first part, then $[\![\varphi_1]\!](S) = [\![\varphi]\!](S)$ for every S.

Now, for the second part we first prove that the "unfolding" does not change semantics, which we do by proving it for each iteration. Consider a stream S, a match M, an $i \in \mathbb{N}$, a CEPL formula ρ, two formulas α_1, α_2 and a valuation ν. We prove that $M \in [\![\rho \texttt{ FILTER } \alpha_1 \wedge \alpha_2]\!](S, i, \nu)$ if, and only if, $M \in [\![(\rho \texttt{ FILTER } \alpha_1) \texttt{ FILTER } \alpha_2]\!](S, i, \nu)$. This is straightforward: $M \in [\![(\rho \texttt{ FILTER } \alpha_1) \texttt{ FILTER } \alpha_2]\!](S, i, \nu)$ holds if $\nu_S \vDash \alpha_1$, $\nu_S \vDash \alpha_2$ and $M \in [\![\rho]\!](S, i, \nu)$, which means the same as $M \in [\![\rho]\!](S, i, \nu)$ and $\nu_S \vDash (\alpha_1 \wedge \alpha_2)$, which is the condition for $M \in [\![\rho \texttt{ FILTER } \alpha_1 \wedge \alpha_2]\!](S, i, \nu)$ to hold. Similarly, we prove that $M \in [\![\rho \texttt{ FILTER } \alpha_1 \vee \alpha_2]\!](S, i, \nu)$ iff $M \in [\![\rho \texttt{ FILTER } \alpha_1 \texttt{ OR } \rho \texttt{ FILTER } \alpha_2]\!](S, i, \nu)$ by definition. $M \in [\![\rho \texttt{ FILTER } \alpha_1 \texttt{ OR } \rho \texttt{ FILTER } \alpha_2]\!](S, i, \nu)$ holds if either $M \in [\![\rho]\!](S, i, \nu)$ and $\nu_S \vDash \alpha_1$ or $M \in [\![\rho]\!](S, i, \nu)$ and $\nu_S \vDash \alpha_2$. This is the same as $M \in [\![\rho]\!](S, i, \nu)$ and either $\nu_S \vDash \alpha_1$ or $\nu_S \vDash \alpha_2$, which is also the same as $M \in [\![\rho]\!](S, i, \nu)$ and $\nu_S \vDash (\alpha_1 \vee \alpha_2)$. Because this is the condition for $M \in [\![\rho \texttt{ FILTER } \alpha_1 \vee \alpha_2]\!](S, i, \nu)$ to hold, then they are equivalent. If we name φ_2 as the result of applying the unfolding, then it follows that $[\![\varphi_2]\!](S) = [\![\varphi_1]\!](S)$ for every S.

Finally, we prove that moving the predicates to their definitions does not affect the semantics either, for which we show that it stays the same after each iteration. Consider a subformula of the form φ' FILTER $p(x)$. The same way as before, we focus on the modified part, i.e., we need to prove that $M \in [\![\varphi' \texttt{ FILTER } p(x)]\!](S, i, \nu)$ iff $M \in [\![\varphi'_p]\!](S, i, \nu)$, where φ'_p is the result of adding the filter $p(x)$ for each definition of x inside φ', i.e., replace R AS x with R AS x FILTER $p(x)$ where R is any relation. First, let S, i, M, ν be any stream, position, match and valuation, respectively, such that $M \in [\![\varphi' \texttt{ FILTER } p(x)]\!](S, i, \nu)$, which means that $\nu_S \vDash p(x)$. We know that, when evaluating every subformula R AS x of φ', the valuation ν must stay the same, because $x \notin \mathrm{bound}(\varphi')$, and thus its definition cannot be inside a +-operator (notice that if it appears inside a +, it represents a value different to x, thus the + subformula can be rewritten using a new variable x'). Similarly to the reasoning above, it holds that for any j, M' and φ', if $\nu'_S \vDash p(x)$, then $M' \in [\![R \texttt{ AS } x \texttt{ FILTER } p(x)]\!](S, j, \nu')$ iff $M' \in [\![R \texttt{ AS } x]\!](S, j, \nu')$. Then, because every subformula R AS x behaves the same, $M \in [\![\varphi'_p]\!](S, i, \nu)$ holds. For the opposite direction, let S, i, M, ν be any stream, position, match and valuation, respectively, such that $M \in [\![\varphi'_p]\!](S, i, \nu)$. We prove that $\nu_S \vDash p(x)$ must hold, thus proving that $M \in [\![\varphi' \texttt{ FILTER } p(x)]\!](S, i, \nu)$ holds by the same argument as above. By contradiction, assume that $\nu_S \vDash \neg p(x)$. Because we showed that when evaluating every R AS x FILTER $p(x)$ in φ'_p, the valuation ν must be the same, the only possible way for $M \in [\![\varphi'_p]\!](S, i, \nu)$ to hold is if all R AS x appear at one side of an OR -operator. However, this would contradict the fact that $x \notin \mathrm{bound}(\varphi')$, thus $\nu_S \vDash p(x)$ must hold, and so must $M \in [\![\varphi' \texttt{ FILTER } p(x)]\!](S, i, \nu)$. Then, φ' FILTER $p(x)$ and φ'_p are equivalent, therefore, if we name φ_{lp} the result of moving all predicates to their definitions, $[\![\varphi_{lp}]\!](S) = [\![\varphi_2]\!](S)$ for every S.

Finally, it is easy to check that the size of φ_{lp} will be at most exponential in the size of φ. Indeed, in each rewriting step (i.e. from φ to φ_1 and from φ_1 to φ_2) we can duplicate the size φ in the worst case. Since the number of rewriting steps are at most linear in the size of φ (if we do the rewriting steps bottom up in the parse tree), we have that $|\varphi_{lp}| \in \mathcal{O}(2^{|\varphi|} \cdot |\varphi|)$.

□

The importance of this result and Theorem 5.2 will become clear in the next sections, where we show that safe formulas in LP-normal form induce efficient evaluation strategies.

6. COMPUTATIONAL MODEL FOR CEP

In this section, we introduce a formal computational model for evaluating CEPL formulas called *match automata*. Similar to classical database management systems (DBMS), it is useful to have a formal model that stands between the query language and the evaluation algorithms, in order to simplify the analysis and optimization of the whole evaluation process. There are several examples of DBMS that are based on this approach like regular expressions and finite state automata (Hopcroft & Ullman, 1979; Aho, 1990), and relational algebra and SQL (Abiteboul et al., 1995; Ramakrishnan & Gehrke, 2003). Here, we propose match automata as the intermediate evaluation model for CEPL and show later how to compile any (unary) CEPL formula into a match automaton.

As its name suggests, match automata (MA) are an extension of *Finite State Automata* (FSA). The first difference from FSA comes from handling streams instead of words. A match automaton is said to run over a stream of tuples, unlike FSA which run over words of a certain alphabet. The second difference arises directly from the first one by the need of processing tuples, which are infinitely many in contrast to the finite input alphabet of FSA. To handle this, our model is extended the same way as a Symbolic Finite Automata (SFA) (Veanes, 2013). SFAs are finite state automata in which the alphabet is described implicitly by a boolean algebra over the symbols. This allows automata to work with a possibly infinite alphabet and, at the same time, use finite state memory for processing the input. Match automata are extended analogously, which is reflected in transitions labeled by (unary) formulas over tuples.

The last difference addresses the need to output matches instead of a boolean answer. A well known extension for FSA are *Finite State Transducers* (Berstel, 2013) which are automata capable of producing an output whenever an input element is read. We follow this idea: MA are allowed to generate and output matches when reading a stream, similar to the class of synchronized transducers (Frougny & Sakarovitch, 1993) (i.e. transducers whose input and output have the same length). Note that, although general transducers have bad decidability properties (Berstel, 2013), the class of synchronized transducers is

closed under union, intersection, and complement, and most of their associated problems are decidable (Frougny & Sakarovitch, 1993). In particular, our model inherit the good properties of synchronized transducers which are exploited in Section 7 for building MA from CEPL formulas.

Before defining the MA model we need some basic definitions. Fix a schema $\mathcal{R}$ and let $\mathbf{F}_u(\mathcal{R})$ be the set of all selection formulas with unary predicates (as defined in Section 5). Given $t \in \text{tuples}(\mathcal{R})$ and $\alpha \in \mathbf{F}_u(\mathcal{R})$, we say that t satisfies α, denoted by $t \models \alpha$, if $\sigma_t \models \alpha$ where σ_t is the function that assigns t to every variable in α (i.e. $\sigma_t(x) = t$ for every $x \in \text{var}(\alpha)$). Finally, without loss of generality we suppose that $\mathbf{F}_u(\mathcal{R})$ contains predicates of the form "$\text{type}(x) = R$" for every $R \in \mathcal{R}$. This will help the automata model to check whether a tuple is of type R or not.

Let $\mathcal{R}$ be a schema and $\bullet, \circ$ be two symbols. A *match automaton* (MA) over $\mathcal{R}$ is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a finite set of states, $\Delta \subseteq Q \times (\mathbf{F}_u(\mathcal{R}) \times \{\bullet, \circ\}) \times Q$ is the transition relation, and $I, F \subseteq Q$ are the set of initial and final states, respectively. Given an $\mathcal{R}$-stream $S = t_0 t_1 \ldots$, a run ρ of $\mathcal{A}$ over S is a sequence of transitions: $\rho : q_0 \xrightarrow{\alpha_0/m_0} q_1 \xrightarrow{\alpha_1/m_1} \cdots \xrightarrow{\alpha_n/m_n} q_{n+1}$ such that $q_0 \in I$, $t_i \models \alpha_i$ and $(q_i, \alpha_i, m_i, q_{i+1}) \in \Delta$ for every $i \leq n$. We say that ρ is *accepting* if $q_{n+1} \in F$ and $m_n = \bullet$. Intuitively, the set of values i such that $m_i = \bullet$ in a run represent the match generated by that run. It is then natural to ask for the last position to be in the match, as otherwise a match could depend on *future* events. We denote by $\text{Run}_n(\mathcal{A}, S)$ the set of accepting runs of $\mathcal{A}$ over S of length n. Further, we denote by $\text{match}(\rho)$ the set of positions where the run *marks* the stream, namely $\text{match}(\rho) = \{i \in [0, n] \mid m_i = \bullet\}$. Given a stream S and $n \in \mathbb{N}$, we define the set of matches of $\mathcal{A}$ over S at position n as: $[\![\mathcal{A}]\!]_n(S) = \{\text{match}(\rho) \mid \rho \in \text{Run}_n(\mathcal{A}, S)\}$ and the set of all matches as $[\![\mathcal{A}]\!](S) = \bigcup_n [\![\mathcal{A}]\!]_n(S)$. Although $[\![\mathcal{A}]\!](S)$ can be an infinite set of matches, $[\![\mathcal{A}]\!]_n(S)$ is always finite.

As an example, consider the match automaton $\mathcal{A}$ depicted in Figure 6.1. In this MA, $\alpha(x) := \text{type}(x) = H$ and $\beta(x) := \text{type}(x) = T \wedge x.temp > 40$. Each transition $\alpha(x) \mid \bullet$ marks one H-tuple and each transition $\beta(x) \mid \bullet$ marks a sequence of T-tuples with

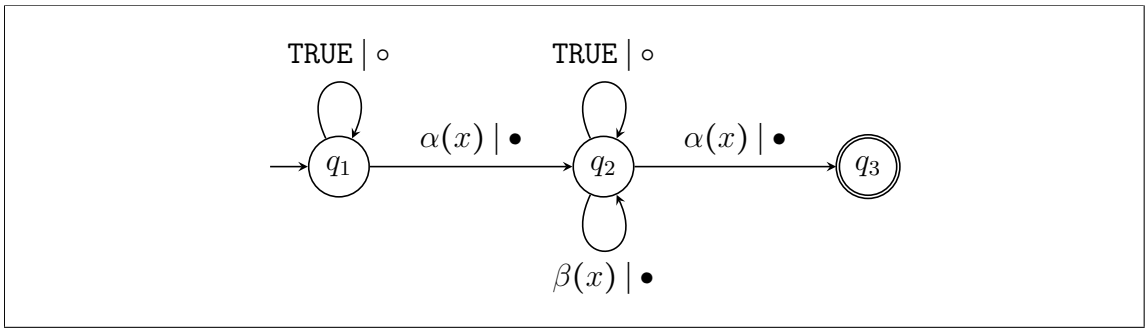

FIGURE 6.1. A match automaton that can generate an unbounded amount of matches over a stream.

temperature bigger than 40. Note also that the transitions labeled by TRUE$|\circ$ allows $\mathcal{A}$ to arbitrarily skip any of the input tuples in the stream. Then, for every stream S, $[\![\mathcal{A}]\!](S)$ represents the set of all matches that begin and end with an H-tuple and also contain some of the T-tuples with temperature higher than 40.

It is important to stress that match automata are designed to be an evaluation model for an expressive sub-fragment of CEPL, called *unary* CEPL (see Section 7 for the formal definition). Several computational models have been proposed for complex event processing (Demers et al., 2006; Pietzuch et al., 2003; Wu et al., 2006; Schultz-Møller et al., 2009); most of them are informal and complex extensions of finite state automata. In our framework, we want to give a step back compared to previous proposals and define a simple but powerful model that captures the *regular core* of CEPL. With "regular" we mean all CEPL formulas that can be checked with finite state memory. Intuitively, formulas like φ_1, φ_2 and φ_3 presented in Section 2 can be evaluated using a bounded amount of memory. In contrast, formula φ_4 needs unbounded memory to store *candidate* events seen in the past, and thus, it calls for a more sophisticated model (e.g. data automata (Segoufin, 2006)). Of course, one would like to have a full-fledged model for CEPL, but this is not possible if we do not understand first its regular core. For these reasons, a computational model for the whole CEP logic is left for future work (see Section 9 for more discussion).

The MA model has good closure properties, for example, under union, intersection, complement and determinization. Formally, we say that a match automaton $\mathcal{A}$ is deterministic if $|I| = 1$ and for any two transitions (p, α_1, m_1, q_1) and (p, α_2, m_2, q_2), either α_1

and α_2 are mutually exclusive (for every t it is not true that $t \vDash \alpha_1$ and $t \vDash \alpha_2$), or $m_1 \neq m_2$ (see (Sakarovitch, 2009) for a similar definition of deterministic letter-to-letter transducers). Then we say that MA are closed under determinization (complement) if for every MA $\mathcal{A}$, there is a deterministic MA $\mathcal{A}^{\text{det}}$ (a MA $\mathcal{A}^{\text{c}}$ resp.) such that for every stream S we have $[\![\mathcal{A}^{\text{det}}]\!](S) = [\![\mathcal{A}]\!](S)$ ($[\![\mathcal{A}^{\text{c}}]\!](S) = \{M \in 2^{\mathbb{N}} \mid M \text{ is finite}\} \setminus [\![\mathcal{A}]\!](S)$ resp.). Furthermore, we say that $\mathcal{A}$ is closed under union (intersection) if for every MA $\mathcal{A}_1$ and $\mathcal{A}_2$, there exists a MA $\mathcal{A}$ such that for every stream S we have $[\![\mathcal{A}]\!](S) = [\![\mathcal{A}_1]\!](S) \cup [\![\mathcal{A}_2]\!](S)$ ($[\![\mathcal{A}]\!](S) = [\![\mathcal{A}_1]\!](S) \cap [\![\mathcal{A}_2]\!](S)$ resp.).

PROPOSITION 6.1. *Match Automata are closed under union, intersection, complement, and determinization.*

PROOF. For the following proof consider any two MA $\mathcal{A}_1 = (Q_1, \Delta_1, I_1, F_1)$, $\mathcal{A}_2 = (Q_2, \Delta_2, I_2, F_2)$ and assume, without loss of generality, that they have disjoint sets of states, i.e., $Q_1 \cap Q_2 = \varnothing$. We first begin by proving closure under union, which is exactly the same as the proof for FSA closure under union. We define the MA $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q, \Delta, I, F)$ as follows. The set of states is $Q = Q_1 \cup Q_2$, the transition relation is $\Delta = \Delta_1 \cup \Delta_2$; the set of initial states is $I = I_1 \cup I_2$ and the set of final states is $F = F_1 \cup F_2$.

Next we prove closure under intersection. We define the MA $\mathcal{A}_1 \cap \mathcal{A}_2 = (Q, \Delta, I, F)$ as follows. The set of states is the Cartesian product $Q = Q_1 \times Q_2$; the transition relation is $\Delta = \{((p_1, p_2), (\alpha_1 \wedge \alpha_2, m), (q_1, q_2)) \mid (p_i, (\alpha_i, m), q_i) \in \Delta_i \text{ for } i \in \{1, 2\}\}$, that is, both conditions α_1 and α_2 must me satisfied by the incoming tuple in order to simulate both transitions with the same mark m from p_1 to q_1 and from p_2 to q_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively; the set of initial states is $I = I_1 \times I_2$ and the set of final states is $F = F_1 \times F_2$.

Now we prove closure under determinization. Define the MA $\mathcal{A}_d = (Q_d, \Delta_d, I_d, F_d)$ component by component. First, the set of states is $Q_d = 2^Q$, that is, each state in Q_d

represents a different subset of Q. Second, the transition relation is:

$$\Delta_d = \{(T, (\alpha, m), U) \mid \alpha \in F\text{-types, and } q \in U \text{ iff there is a } p \in T \text{ and } \alpha' \in \mathbf{F}_u(\mathcal{R}) \text{ such that } (p, (\alpha', m), q) \in \Delta \text{ and } \alpha \vDash \alpha'\}.$$

Here, F is the set of all formulas in the transitions of Δ and we use the notion of F-types defined in the proof of Theorem 7.3. Finally, the sets of initial and final states are $I_d = \{I\}$ and $F_d = \{T \mid T \in Q_d \wedge T \cap F \neq \varnothing\}$. The key notion here is the one of F-types, which partitions the set of all tuples in a way that if a tuple t satisfies a formula $\alpha_t \in F$-types, then α_t implies the conditions of all transition that a run of $\mathcal{A}$ could take when reading t. This allows us to then apply a determinization algorithm similar to the one for FSA. Notice that $\alpha_1 \vDash \neg\alpha_2$ for every two different formulas $\alpha_1, \alpha_2 \in F$-types, so the resulting MA $\mathcal{A}_d$ is deterministic.

Finally, we prove closure under complementation. Basically, the complementation of a MA is no more than determinizing it and complementing the set of final states. Formally, we define the MA $\mathcal{A}_1^c = (Q, \Delta, I, F)$ as follows. Consider the deterministic MA $\det(\mathcal{A}_1) = (Q_d, \Delta_d, I_d, F_d)$. Then, the set of states, the transition relation and the set of initial states are the same as of $\det(\mathcal{A}_1)$, i.e., $Q = Q_d$, $\Delta = \Delta_d$ and $I = I_d$, and the set of final states is $F = Q \setminus F_d$. □

A reasonable question to ask at this point is whether one can efficiently evaluate a MA over a stream. The notion of efficiency here is challenging since we would like to compute matches in one pass and using a restricted amount of resources. Streaming algorithms (Ikonomovska & Zelke, 2013; Golab & Özsu, 2003) are a natural example of efficiency, as they usually restrict the time allowed to process each tuple (e.g., linear in the size of the tuple) and the space needed to process the first n items of a stream (e.g., sublinear in n). Since we want to output matches, we cannot expect to use less than linear space in the processed data (a match could be as long as the stream itself). Another problem for defining the concept of efficiency is that the input object (a stream) is infinite. For this reason, we associate to a stream S a special instruction $\texttt{yield}_S$ that

returns the next element of the stream S. Then, we say that an *efficient evaluation strategy* for a match automaton $\mathcal{A}$ is an algorithm such that for every stream S (1) the update time between two calls to $\texttt{yield}_S$ is bounded by $\mathcal{O}(|t|)$, where t is the tuple returned by the first of such calls, and (2) it maintains a data structure D such that, after calling $\texttt{yield}_S$ n times, the set of matches $[\![\mathcal{A}]\!]_n(S)$ can be enumerated from D with constant delay. The latter condition basically imposes that no processing is done during output generation. Formally, it requires the existence of a routine `enumerate` that receives D as input and outputs all matches in $[\![\mathcal{A}]\!]_n(S)$ without repetitions, while spending a constant amount of time before and after each output. The requirement (1) is a natural restriction imposed in the streaming literature (Ikonomovska & Zelke, 2013), while (2) is the minimum that we can ask if an arbitrarily large set of outputs must be produced (Bagan, Durand, & Grandjean, 2007). We stress that the notion of efficiency introduced here considers the *data complexity* of the problem, namely, the number of states and transitions of $\mathcal{A}$ are considered constant in the asymptotic analysis. Notice also that if the schema is fixed and the values use a fixed amount of memory, we can process each tuple in constant time (again, in data complexity).

The tools for efficiently evaluating every match automaton have not been developed yet: this is an interesting research direction to pursue for CEPL query evaluation. Instead, we focus on the evaluation of a subclass of MA, called *finitely ambiguous* MA. Formally, for $K \in \mathbb{N}$ we say that a MA $\mathcal{A}$ is *K-ambiguous* if for every stream S and for every $n \geq 0$ it holds that $|\mathrm{Run}_n(\mathcal{A}, S)| \leq K$. Further, we say that $\mathcal{A}$ is finitely ambiguous if $\mathcal{A}$ is K-ambiguous for some $K \in \mathbb{N}$. Finitely ambiguous automata have been extensively studied in the past (Sakarovitch, 2009; Seidl, 1990) with very interesting applications (Mohri, 1997).

Theorem 6.1. *Every finitely ambiguous MA $\mathcal{A}$ has an efficient evaluation strategy.*

PROOF. Consider a K-ambiguous MA $\mathcal{A} = (Q, \Delta, I, F)$ and the procedure EVAL$[\mathcal{A}]$ in Algorithm 1 as an efficient evaluation strategy of $\mathcal{A}$ over any stream $S = t_0 t_1 \cdots$. The

procedure basically iterates over all tuples of S in the while clause (line 3), assigning at each iteration the corresponding tuple to the variable t with the instruction $\texttt{yield}_S$.

Algorithm 1 Evaluate $\mathcal{A} = (Q, \Delta, I, F)$ over a stream S

```
1:  procedure EVAL[A](S)
2:      E ← {(q, ∅) | q ∈ I}
3:      while t ← yield_S do
4:          E' ← ∅
5:          for all (q, M) ∈ E do
6:              for all (q, α, m, q') ∈ Δ do
7:                  if t ⊨ α ∧ m = • then
8:                      E' ← E' ∪ { (q', M ∪ {i}) }
9:                  else if t ⊨ α ∧ m = ∘ then
10:                     E' ← E' ∪ { (q', M) }
11:                 end if
12:             end for
13:         end for
14:         E ← E'
15:         enumerate({M | ∃q ∈ F. (q, M) ∈ E})
16:     end while
17: end procedure
```

Now we analyze the efficiency of the algorithm regarding the restrictions imposed in Section 6. At every iteration i (for every $i \geq 0$) the set E keeps track of all runs of $\mathcal{A}$ over the prefix $t_0 \dots t_i$ of S, where each run ρ is represented by a pair (q, M) such that q is the last state of ρ and $M = \text{match}(\rho)$. Then in line 15 it calls the subprocedure `enumerate` for returning all matches from runs that ends in a final state. As stated in Section 6, this is done by a different process, and does not interfere with the asymptotic analysis.

We suppose that M is modeled as a linked list, therefore adding a new position $M \cup \{i\}$ takes constant time (line 8). Moreover, duplicating M (e.g. when we have a non-deterministic branch in $\mathcal{A}$) would also take constant time if we consider that the data (positions) of all the matches is stored together in a tree-like structure which would allow runs to share duplicated sections between them. Then, the pair (q, M) could be stored as a pair (q, n) where n is a node, and M is determined by the positions from n to the root. The verification of the **if** conditions (line 7 and 9) requires reading t and verifying whether $t \vDash \alpha$ which can be done in constant time (see this assumption in Section 3.1), thus it

takes $\mathcal{O}(|t|)$ in total. Then, because the **for all** clauses (line 6) iterates over all transitions, it clearly takes time $\mathcal{O}(|\mathcal{A}| \cdot |t|)$. Now, it is straightforward to check that K-ambiguity implies that, for every prefix $t_0 \dots t_i$ of S, the number of runs of $\mathcal{A}$ over S that end at the same state cannot be more than K, i.e., $|\{(q, M) \in E\}| \leq K$ for every $q \in Q$. Indeed, if this does not happen, one could extend all runs ending in this state to have strictly more than K accepting runs, contradicting the fact that $\mathcal{A}$ is K-ambiguous. Therefore, we conclude that the size of E is always bounded by $K \cdot |\mathcal{A}|$ and the **for all** clause in line 5 iterates over at most $K \cdot |\mathcal{A}|$ pairs taking time at most $\mathcal{O}(K \cdot |\mathcal{A}|^2 \cdot |t|)$ in total. Recall that we are considering the data complexity of the problem which means that $K \cdot |\mathcal{A}|^2$ is a constant factor and, thus, the procedure EVAL$[\mathcal{A}]$ takes time $\mathcal{O}(|t|)$ between each call to $\texttt{yield}_S$. In other words, EVAL$[\mathcal{A}]$ satisfies the first condition of an efficient evaluation strategy.

Finally, one can easily see that EVAL$[\mathcal{A}]$ also satisfies the second condition of an efficient evaluation strategy. Indeed, the matches M are linked lists of pointers and, therefore, each time that `enumerate` has to output the set of matches, it only needs to follow the linked lists taking constant delay in the whole process. □

Even though the previous results seems restricted to a narrow subclass of MA, we will see next that it allows for efficient evaluation of an important fragment of CEPL.

7. COMPILING UNARY CEPL INTO MATCH AUTOMATA

For evaluating unary CEPL we follow an automata-based strategy. Specifically, we show how to compile a (unary) CEPL formula φ into an equivalent MA $\mathcal{A}_\varphi$, meaning that $[\![\varphi]\!](S) = [\![\mathcal{A}_\varphi]\!](S)$ for every stream S, to later evaluate $\mathcal{A}_\varphi$ over streams. Although we do not know how to efficiently evaluate MA in general, we identify that an interesting fragment of (unary) CEPL can be evaluated efficiently.

As previously mentioned, we study here how to compile a subfragment of CEPL that we call *unary* CEPL. Formally, we say that a CEPL formula φ is unary if, for every subformula of φ of the form φ' `FILTER` α, all predicates of α are unary (i.e. $\alpha \in \mathbf{F}_u(\mathcal{R})$). For example, formulas φ_1, φ_2, and φ_3 in Section 2 are unary, but formula φ_4 is not (the predicate $y.id = x.id$ is binary). It is important to mention that, although the unary fragment seems restricted, it already presents non-trivial computational challenges. The evaluation of full CEPL is an interesting project on its own, since it requires new insights on rewriting techniques and more powerful computational models featuring translations and efficient evaluation strategies. We leave this direction for future work.

We start by presenting the compilation of unary core CEPL into MA. This construction is intimately related with the safeness condition and LP-normal form (see Section 5).

Theorem 7.1. *For every formula φ in unary core-CEPL, there exists a MA $\mathcal{A}_\varphi$ equivalent to φ. Furthermore, $\mathcal{A}_\varphi$ is of size at most linear in $|\varphi|$ if φ is safe and in LP-normal form, and of size at most double exponential in $|\varphi|$ otherwise.*

PROOF. For the sake of simplicity, for this proof we will add to the model of MA the ability to have ϵ-transitions. Formally, now a transition relation has the structure $\Delta \subseteq Q \times ((\mathbf{F}_u(\mathcal{R}) \times \{\bullet, \circ\}) \cup \{\epsilon\}) \times Q$. This basically means the automaton can have transitions of the form (p, ϵ, q) that can be part of a run and, if so, the automaton passes from state p to q without reading nor marking any new tuple. This does not give any additional power to MA, since any ϵ-transition (p, ϵ, q) can be removed by adding, for each incoming transition

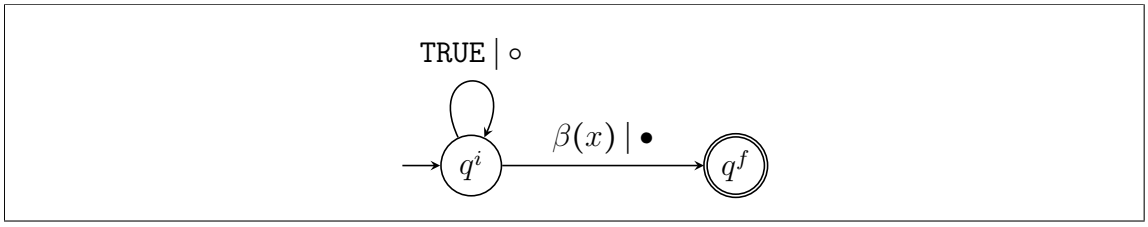

FIGURE 7.1. A match automaton for the formula R AS x FILTER $\alpha(x)$. Here, $\beta(x) = (\text{type}(x) = R) \wedge \alpha(x)$.

of p, an equivalent incoming one to q, and for each outgoing transition of q an equivalent outgoing one from p.

The result of Theorem 5.3 shows that we can rewrite every core CEPL formula as a formula in LP-normal form, so we consider that, if φ is not in LP-normal form, then it is first translated into one with an exponential growth from the beginning. We now give a construction that, for every core CEPL formula φ in LP-normal form, defines a MA $\mathcal{A}$ such that for every match M, $M \in [\![\mathcal{A}]\!](S_i)$ iff there exists a valuation ν such that $M \in [\![\varphi]\!](S, i, \nu)$ (recall that S_i is the stream $t_i t_{i+1} \ldots$). Moreover, we show two properties: (1) for every accepting run ρ there exists a valuation ν such that every $x \in \text{dom}(\nu)$ appears exactly once in ρ and only at the transition $\nu(x)$ of ρ; (2) for every ν there exists an accepting run ρ of $\mathcal{A}$ over S_i such that every $x \in \text{vdef}_+(\varphi)$ that appears in a transition of ρ appears while reading $S[\nu(x)]$. This construction is done recursively in a bottom-up fashion such that, for every subformula, an equivalent MA is built from the MA of its subformulas. Let ψ be any subformula of φ. Then, the MA $\mathcal{A}$ is defined as follows:

- If $\psi = R$ AS x FILTER $\alpha(x)$ then $\mathcal{A} = (Q, \Delta, \{q^i\}, \{q^f\})$ with the set of states $Q = \{q^i, q^f\}$ and the transitions $\Delta = \{(q^i, (\texttt{TRUE}, \circ), q^i), (q^i, (\beta(x), \bullet), q^f)\}$, where $\beta(x) = (\text{type}(x) = R) \wedge \alpha(x)$. Graphically, the automaton is the one in figure 7.1. If ψ has no FILTER the automaton is the same but with $\beta(x) = (\text{type}(x) = R)$.
- If $\psi = \psi_1$ OR ψ_2 and $\mathcal{A}_1 = (Q_1, \Delta_1, \{q_1^i\}, \{q_1^f\})$ and $\mathcal{A}_2 = (Q_2, \Delta_2, \{q_2^i\}, \{q_2^f\})$ are the automata for ψ_1 and ψ_2,respectively, then $\mathcal{A} = (Q, \Delta, \{q^i\}, \{q^f\})$ where Q is the union of the states of $\mathcal{A}_1$ and $\mathcal{A}_2$ plus the new initial and final states q^i, q^f, and Δ is the union of Δ_1 and Δ_2 plus the empty transitions from q^i to the initial states of $\mathcal{A}_1$ and $\mathcal{A}_2$,

and from the final states of $\mathcal{A}_1$ and $\mathcal{A}_2$ to q^f. Formally, $Q = Q_1 \cup Q_2 \cup \{q^i, q^f\}$ and $\Delta = \Delta_1 \cup \Delta_2 \cup \{(q^i, \epsilon, q_1^i), (q^i, \epsilon, q_2^i), (q_1^f, \epsilon, q^f), (q_2^f, \epsilon, q^f)\}$.

- If $\psi = \psi_1 \,;\, \psi_2$ and $\mathcal{A}_1 = (Q_1, \Delta_1, \{q_1^i\}, \{q_1^f\})$ and $\mathcal{A}_2 = (Q_2, \Delta_2, \{q_2^i\}, \{q_2^f\})$ are the automata for ψ_1 and ψ_2, respectively, we first define $X = X_1 \cap X_2$, where $X_i = \mathrm{vdef}_+(\rho_i)$, and define $\mathcal{X} = 2^X$. Now we define the components of $\mathcal{A} = (Q, \Delta, \{q_1^i\}, \{q_2^f\})$. The set of states is $Q = (Q_1 \cup Q_2) \times \mathcal{X}$. Then, the transition relation consists of three parts. The first is $\Delta_1' = \{((p, Y), (\alpha, m), (q, Y')) \mid (p, (\alpha, m), q) \in \Delta_1 \text{ and } Y' = Y \cup (\mathrm{var}(\alpha) \cap X)\}$, which allows $\mathcal{A}$ to simulate $\mathcal{A}_1$, gathering the variables of interest at each transition. The second part is $\Delta_2' = \{((p, Y), (\alpha, m), (q, Y)) \mid (p, (\alpha, m), q) \in \Delta_2 \text{ and } Y \cap \mathrm{var}(\alpha) = \varnothing\}$, which allows $\mathcal{A}$ to simulate $\mathcal{A}_2$ restricting that no variable of interest can be seen again. The third part is $\Delta_3' = \{((q_1^f, Y), \epsilon, (q_2^i, Y)) \mid Y \in \mathcal{X}\}$, which ends the simulation of $\mathcal{A}_1$ and begins the one of $\mathcal{A}_2$. Then, the transition relation Δ is defined as $\Delta = \Delta_1' \cup \Delta_2' \cup \Delta_3'$.
- If $\psi = \psi_1+$ and $\mathcal{A}_1 = (Q_1, \Delta_1, \{q_1^i\}, \{q_1^f\})$ is the automaton for ψ_1, then $\mathcal{A}$ is defined as $(Q, \Delta, \{q_1^i\}, \{q_1^f\})$ where $Q = Q_1$ and $\Delta = \Delta_1 \cup \{(q_1^f, \epsilon, q_1^i)\}$. Basically, is the same automaton for ψ_1 with an ϵ-transition from the final to the initial state.

Now, we need to prove that the previous construction satisfies Theorem 7.1. We will prove this by induction over the subformulas of φ, i.e., assume as induction hypothesis that the theorem holds for any subformula ψ and its respective MA $\mathcal{A}$.

First, consider the base case $\psi = R \texttt{ AS } x \texttt{ FILTER } \alpha$. If $M \in [\![\mathcal{A}]\!](S_i)$ then there is a run ρ that gets to the accepting run such that $\mathrm{match}(\rho) = M$. Moreover, ρ must pass through the transition $(q^i, (\mathrm{type}(x) = R \wedge \alpha(x), q^f)$ while reading a tuple t_j at some position $j \geq i$. Then, consider the valuation $\nu : x \to j$. Clearly, $M = \{\nu(x)\}$, $\mathrm{type}(t_j) = R$, and $\nu_S \vDash \alpha$, thus $M \in [\![\psi]\!](S, i, \nu)$. Further, notice that property (1) holds. For the other direction, consider that $M \in [\![\psi]\!](S, i, \nu)$ for some valuation ν. Then M must contain only position $m = \nu(x)$ such that $\mathrm{type}(S[m]) = R$, $i \leq m$ and $\nu_S \vDash \alpha$ hold. Then $\rho = (q^i, (\texttt{TRUE}, \circ), q^i)^{m-i} \cdot (q^i, (\beta(x), \bullet), q^f)$ is an accepting run of $\mathcal{A}$ over S_i, where $(q^i, (\texttt{TRUE}, \circ), q^i)^{m-i}$ means that it takes the initial loop transition $m - i$ times. Because $\mathrm{match}(\rho) = \{m\} = M$, then $M \in [\![\mathcal{A}]\!](S_i)$. Moreover, notice that property (2) holds.

Now, consider the case $\psi = \psi_1 \texttt{ OR } \psi_2$. If $M \in [\![\mathcal{A}]\!](S_i)$, then there is an accepting run ρ that also represents either an accepting run of $\mathcal{A}_1$ or $\mathcal{A}_2$ (removing the ϵ transitions at the beginning and end). Assume without loss of generality that it is the first case. Then, by induction hypothesis, there is a valuation ν such that $M \in [\![\psi_1]\!](S, i, \nu)$. By definition this means that $M \in [\![\psi]\!](S, i, \nu)$. Notice that hy induction hypothesis, property (1) holds. For the other direction, consider that $M \in [\![\psi]\!](S, i, \nu)$ for some valuation ν. Then, either $M \in [\![\psi_1]\!](S, i, \nu)$ or $M \in [\![\psi_2]\!](S, i, \nu)$ holds. Without loss of generality, consider the former case. By induction hypothesis, it means that $M \in [\![\mathcal{A}_1]\!](S_i)$, so there is an accepting run ρ' of $\mathcal{A}_1$ over S_i such that $\mathrm{match}(\rho') = M$. Because Δ contains Δ_1 then the run $\rho = (q^i, \epsilon, q_1^i) \cdot \rho' \cdot (q_1^f, \epsilon, q^f)$ is an accepting run of $\mathcal{A}$ over S_i. Furthermore, by induction hypothesis property (2) holds.

Next, consider the case $\psi = \psi_1 \,;\, \psi_2$. If $M \in [\![\mathcal{A}]\!](S_i)$, then there is an accepting run ρ of the form $\rho : \rho_1 \cdot ((q_1^f, Y), \epsilon, (q_2^i, Y)) \cdot \rho_2$ and, because of the construction, $M_1 = \mathrm{match}(\rho_1) \in [\![\mathcal{A}_1]\!](S_i)$ and $M_1 = \mathrm{match}(\rho_1) \in [\![\mathcal{A}_1]\!](S_j)$, with $j = \max(M_1) + 1$. Then by induction hypothesis there are valuations ν_1 and ν_2 such that $M_1 \in [\![\psi_1]\!](S, i, \nu_1)$, $M_2 \in [\![\psi_2]\!](S, j, \nu_2)$. Moreover, because all transitions of ρ_1 are before the ones of ρ_2 and because of property (1), $\mathrm{dom}(\nu_1) \cap \mathrm{dom}(\nu_2) = \varnothing$. Therefore, we can define ν such that $\nu(x) = \nu_1(x)$ if $x \in \mathrm{dom}(\nu_1)$ and $\nu(x) = \nu_2(x)$ if $x \in \mathrm{dom}(\nu_2)$. Clearly, because ν represents both ν_1 and ν_2, it holds that $M \in [\![\psi]\!](S, i, \nu)$. Moreover, because of the construction, no $x \in X$ can appear twice in ρ, and because ν_1 and ν_2 satisfy property (1), so does ν. For the other direction, consider a match M such that $M \in [\![\psi]\!](S, i, \nu)$ for some valuation ν. Then there exist matches M_1 and M_2 such that $M_1 \in [\![\psi_1]\!](S, i, \nu)$, $M_2 \in [\![\psi_2]\!](S, j, \nu)$ and $M = M_1 \cdot M_2$, where $j = \max(M_1) + 1$. By induction hypothesis, there exist accepting runs ρ_1 and ρ_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that $\mathrm{match}(\rho_1) = M_1$, $\mathrm{match}(\rho_2) = M_2$. Moreover, from property (2) we know that every x that appears in a transition of ρ_i appears while reading $S[\nu(x)]$. In particular, every $x \in X$ that appears in ρ_1 cannot appear in ρ_2 because all transitions of ρ_1 are before the ones of ρ_2, so they have no positions in common. Therefore, the run ρ of $\mathcal{A}$ that simulates ρ_1 ends at a state (q_1^f, Y) (and position j) such that no $x \in Y$ is in ρ_2, thus ρ can continue by simulating ρ_2 and reaching a final state,

thus ρ is an accepting run. Notice that $\mathrm{match}(\rho) = M$, thus $M \in [\![\mathcal{A}]\!](S_i)$. Moreover, ρ clearly satisfies property (2).

Finally, consider the case $\psi = \psi_1+$. If $M \in [\![\mathcal{A}]\!](S_i)$, it means that there is an accepting run ρ of $\mathcal{A}$ over S_i. We define k to be the number of times that ρ passes through the final state q^f, and prove by induction over k that $M \in [\![\psi]\!](S, i, \nu)$. If $k = 1$, it means that ρ is also an accepting run of $\mathcal{A}_1$, thus $M \in [\![\mathcal{A}_1]\!](S_i)$ and, by (the first) induction hypothesis, there exists some valuation ν such that $M \in [\![\psi_1]\!](S, i, \nu)$, which implies $M \in [\![\psi]\!](S, i, \nu)$. Now, consider the case $k > 1$. It means that ρ has the form $\rho = \rho_1 \cdot (q^f, \epsilon, q^i) \cdot \rho_2$ where ρ_2 passes through q^f $k - 1$ times. Then, $M_1 = \mathrm{match}(\rho_1)$ is an accepting run of $\mathcal{A}_1$ and thus $M_1 \in [\![\psi_1]\!](S, i, \nu)$ for some ν. Furthermore, ρ_2 is an accepting run of $\mathcal{A}$, thus if $M_2 = \mathrm{match}(\rho_2)$ then by induction hypothesis $M_2 \in [\![\psi]\!](S, j, \nu)$ for some ν, where $j = \max(M_1)$. If $M = M_1 \cdot M_2$ then $M \in [\![\psi_1 \,;\, \psi_1+]\!](S, i, \nu)$, thus $M \in [\![\psi]\!](S, i, \nu)$. Notice that a valuation ν' such that $\mathrm{dom}(\nu') = \varnothing$ also satisfies $M \in [\![\psi]\!](S, i, \nu')$, thus it trivially satisfies property (1). For the other direction, consider a match M such that $M \in [\![\psi]\!](S, i, \nu)$ for some valuation ν. Then there exists ν' such that either $M \in [\![\psi_1]\!](S, i, \nu[\nu' \to U])$ or $M \in [\![\psi_1 \,;\, \psi_1+]\!](S, i, \nu[\nu' \to U])$ where $U = \mathrm{vdef}_+(\psi_1)$. We now prove, by induction over the number of iterations, that $M \in [\![\mathcal{A}]\!](S_i)$. If there is just one iteration, then $M \in [\![\psi_1]\!](S, i, \nu[\nu' \to U])$ and, by induction hypothesis, $M \in [\![\mathcal{A}_1]\!](S_i)$, so there is an accepting run ρ of $\mathcal{A}_1$ over S_i such that $\mathrm{match}(\rho) = M$. Because $\Delta_1 \subseteq \Delta$, then ρ is also an accepting run of $\mathcal{A}$, thus $M \in [\![\mathcal{A}]\!](S_i)$. If there are k iterations with $k > 1$, it means that $M \in [\![\psi_1 \,;\, \psi_1+]\!](S, i, \nu[\nu' \to U])$. Therefore, there exist matches M_1 and M_2 such that $M_1 \in [\![\psi_1]\!](S, i, \sigma[\sigma' \to U])$, $M_2 \in [\![\psi_1+]\!](S, j, \nu[\nu' \to U])$ and $M = M_1 \cdot M_2$, where $j = \max(M_1)$. Then, by induction hypothesis, there exist accepting runs ρ_1 of $\mathcal{A}_1$ over S and ρ_2 of $\mathcal{A}$ over S_j such that $\mathrm{match}(\rho_1) = M_1$ and $\mathrm{match}(\rho_2) = M_2$ and, because $\Delta_1 \subseteq \Delta$, ρ_1 is also an accepting run of $\mathcal{A}$. Then, the run $\rho = \rho_1 \cdot (q^f, \epsilon, q^i) \cdot \rho_2$ is an accepting run of $\mathcal{A}$ over S. Furthermore, $\mathrm{match}(\rho) = M_1 \cdot M_2 = M$ thus $M \in [\![\mathcal{A}]\!](S_i)$. Notice that $\mathrm{vdef}_+(\psi) = \varnothing$, thus every run satisfies property (2).

Finally, it is clear that the size of $\mathcal{A}$ is linear with respect to the size of φ if φ is safe and already in LP-normal form. However, if φ is not safe, then the construction for

; have an exponential blow-up. Furthermore, if φ is not in LP-normal form, then it is first translated into an equivalent CEPL formula φ' that is, adding an exponential growth. Then, $\mathcal{A}_\varphi$ is of size at most double exponential in $|\varphi|$ and of size at most linear if φ is safe and in LP-normal form. □

Next we focus on how to construct MA from formulas with extended operators like `AND`, `ALL`, and `UNLESS`. In contrast to the core fragment, these operators are more complicated to evaluate because of the size of their respective MAs. More specifically, let φ_1 and φ_2 be two unary ecore-CEPL formulas and $X = \mathrm{vdef}_+(\varphi_1) \cap \mathrm{vdef}_+(\varphi_2)$.

Theorem 7.2. *Let $\varphi = \varphi_1$ `OP` φ_2 be a CEPL formula with `OP` $\in \{$`AND`, `ALL`, `UNLESS`$\}$, and let $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ be two MA equivalent to φ_1 and φ_2, respectively. Then, there is a MA $\mathcal{A}_\varphi$ equivalent to φ of size at most $\mathcal{O}(|\mathcal{A}_{\varphi_1}| \cdot |\mathcal{A}_{\varphi_2}| \cdot 2^{|X|})$ if `OP` $\in \{$`AND`, `ALL`$\}$ and at most $\mathcal{O}(|\mathcal{A}_{\varphi_1}| \cdot 2^{|\mathcal{A}_{\varphi_2}|})$ if `OP` = `UNLESS`.*

PROOF. For each `OP` $\in \{$`AND`, `ALL`, `UNLESS`$\}$ we give a construction for a MA $\mathcal{A} = (Q, \Delta, I, F)$ that is equivalent to the CEPL formula φ of the form φ_1 `OP` φ_2. Moreover, we prove that properties (1) and (2) stated in Theorem 7.1 still hold. Let $\mathcal{A}_1 = (Q_1, \Delta_1, I_1, F_1)$ and $\mathcal{A}_2 = (Q_2, \Delta_2, I_2, F_2)$ be the MA equivalent to φ_1 and φ_2 respectively, define $X = X_1 \cap X_2$, where $X_i = \mathrm{vdef}_+(\varphi_i)$, and define $\mathcal{X} = 2^X$. For the first the case `OP` = `AND`, the automaton $\mathcal{A}$ is defined as follows. The set of states is $Q = Q_1 \times Q_2 \times \mathcal{X}$. Then the set transition relation consists of all transitions of the form $((p_1, p_2, Y), (\alpha_1 \wedge \alpha_2, m), (q_1, q_2, Y')$ such that there are transitions $(p_1, (\alpha_1, m), q_1) \in \Delta_1$ and $(p_2, (\alpha_2, m), q_2) \in \Delta_2$ with $\mathrm{var}(\alpha_1) \cap Y = \mathrm{var}(\alpha_2) \cap Y = \varnothing$ and $Y' = Y \cup (X \cap (\mathrm{var}(\alpha_1) \cup \mathrm{var}(\alpha_2)))$. Finally, the sets of initial and final states are $I = I_1 \times I_2 \times \{\varnothing\}$ and $F = F_1 \times F_2 \times \mathcal{X}$, respectively. Basically, the automaton simulates both $\mathcal{A}_1$ and $\mathcal{A}_2$ and force them to mark positions at the same time. Moreover, if there is a variable in both φ_1 and φ_2, it force them to mark it at the same position.

Now we define the automaton $\mathcal{A}$ for the case `OP` = `ALL`. Basically, the automaton simulates both $\mathcal{A}_1$ and $\mathcal{A}_2$, allowing them to stop their runs at different positions of the

stream. For this, we add a new symbol $\perp$. If a run of $\mathcal{A}$ is at state (q_1, q_2, Y) it means that the simulations are in q_1 and q_2, respectively, and that the variables of X seen so far are Y. Moreover, if state is $\perp$, it means that the simulation of that automaton has ended. The set of states Q is defined as $Q = (Q_1 \cup \{\perp\}) \times (Q_2 \cup \{\perp\}) \times \mathcal{X}$. Then, to simulate both automata simultaneously we add all transitions of the form $((p_1, p_2, Y), (\alpha_1 \wedge \alpha_2, m), (q_1, q_2, Y'))$ such that there are transitions $(p_1, (\alpha_1, m_1), q_1) \in \Delta_1$ and $(p_2, (\alpha_2, m_2), q_2) \in \Delta_2$ with $\mathrm{var}(\alpha_1) \cap Y = \mathrm{var}(\alpha_2) \cap Y = \varnothing$, $Y' = Y \cup (X \cap (\mathrm{var}(\alpha_1) \cup \mathrm{var}(\alpha_2)))$, and $m = \bullet$ if either $m_1 = \bullet$ or $m_2 = \bullet$, and $m = \circ$ otherwise. Moreover, to continue the simulation of the first MA after the other ended, we add transitions $((p_1, \perp, Y), (\alpha_1, m_1), (q_1, \perp, Y'))$ such that $(p_1, (\alpha_1, m_1), q_1) \in \Delta_1$, $\mathrm{var}(\alpha_1) \cap Y = \varnothing$ and $Y' = Y \cup (\mathrm{var}(\alpha_1) \cap X)$. Similarly for the second MA, we add transitions $((\perp, p_2, Y), (\alpha_2, m_2), (\perp, q_2, Y'))$ such that $(p_2, (\alpha_2, m_2), q_2) \in \Delta_2$, $\mathrm{var}(\alpha_2) \cap Y = \varnothing$ and $Y' = Y \cup (\mathrm{var}(\alpha_2) \cap X)$. Finally, to allow one of the automata to end its simulation, we add the ϵ-transitions $((p_1, p_2^f, Y), \epsilon, (p_1, \perp, Y))$ and $((p_1^f, p_2, Y), \epsilon, (\perp, p_2, Y))$, where $p_1^f \in F_1$, $p_2^f \in F_2$. The set of initial states is defined as $I = I_1 \times I_2 \times \{\varnothing\}$ and the set of final states as $F = \{(q_1, q_2, Y) \mid q_1 \in F_1 \cup \{\perp\} \wedge q_2 \in F_2 \cup \{\perp\} \wedge Y \in \mathcal{X}\}$.

Now, we define $\mathcal{A}$ for the case `OP = UNLESS`. For this part, we need to define some automaton transformations. Let $\mathcal{A}_2^d = (Q_2^d, \Delta_2^d, I_2^d, F_2^d)$ be the result of applying to $\mathcal{A}_2$ the determinization construction in Proposition 6.1. Let $\mathcal{A}_2' = (Q_2', \Delta_2', I_2', F_2')$ be a boolean automaton for $\mathcal{A}_2$ which is essentially an automaton that gets to an accepting state if there has been a match in $\mathcal{A}_2$ in the prefix read until that point. Its structure is the same, except that it has an extra state q_a which is the only accepting one, i.e., $Q_2' = Q_2 \cup \{q_a\}$, $I_2' = I_2$ and $F = \{q_a\}$. Because $\mathcal{A}_2'$ does not return a match, all transitions of Δ_2 are copied without their matching symbols, i.e., for every transition $(p, (\alpha, m), q) \in \Delta_2$, the transition (p, α, q) is added in Δ_2'. In addition, for every transition in Δ_2 of the form $(p, (\alpha, \bullet), q)$ with $q \in F_2$, it adds the transition (p, α, q_a). We consider only $\bullet$-transitions because the MA semantics restricts that the last transition of a run must be with $\bullet$. With this construction we claim that when we run $\mathcal{A}_2'$ over a stream S, if a run is at the accepting state q_a while reading position i, then there is a match $M \in [\![\mathcal{A}_2]\!](S)$ such that $\max(M) \leq i$. Now, let $\mathcal{A}_2^b$

be the result of applying determinization over $\mathcal{A}'_2$, which is the automaton that we will use in the sequel. Notice that for both $\mathcal{A}^d_2$ and $\mathcal{A}^b_2$, the states can be seen as subsets of Q_2 because of the determinization part of their constructions. Because of this, we will identify states of Q^d_2 and Q^b_2 as $\{q^d_1, q^d_2, \ldots, q^d_k\}$ and $\{q^b_1, q^b_2, \ldots, q^b_k\}$ respectively, where both are associated to the subset $\{q_1, q_2, \ldots, q_k\} \subseteq Q_2$. Now we are ready to define $\mathcal{A}$. The set of states is defined as $Q = (Q_1 \times Q^d_2) \cup (Q_1 \times Q^b_2)$. As notation, we write the states of Q^d_2 and Q^b_2 as $\mathbf{q}$ to illustrate that it is associated to a subset of states (of Q_2). We define the transition relation as follows. First, for every $(p_1, (\alpha_1, \circ), q_1) \in \Delta_1$ and $(\mathbf{p_2}, (\alpha_2, \circ), \mathbf{q_2}) \in \Delta^d_2$ we add $((p_1, \mathbf{p_2}), (\alpha_1 \wedge \alpha_2, \circ), (q_1, \mathbf{q_2}))$ into Δ. Second, for every $(p_1, (\alpha_1, \circ), q_1) \in \Delta_1$ and $(\mathbf{p_2}, \alpha_2, \mathbf{q_2}) \in \Delta^b_2$ such that $\mathbf{q_2} \notin F^b_2$, we add $((p_1, \mathbf{p_2}), (\alpha_1 \wedge \alpha_2, \circ), (q_1, \mathbf{q_2}))$ into Δ. Finally, for every $(p_1, (\alpha_1, \bullet), q_1) \in \Delta_1$, $(\mathbf{p_2}, \alpha_2, \mathbf{q_2}) \in \Delta^b_2$ we add $((p_1, \mathbf{p'_2}), (\alpha_1 \wedge \alpha_2, \bullet), (q_1, \mathbf{q_2}))$ into Δ, where $\mathbf{p'_2}$ is the analogous of $\mathbf{p_2}$ in $\mathcal{A}^d_2$, i.e., if $\mathbf{p_2} = \{p^b_1, p^b_2, \ldots, p^b_k\}$, then $\mathbf{p'_2} = \{p^d_1, p^d_2, \ldots, p^d_k\}$. The set of initial states is defined as $I = I_1 \times I^d_2$ and the set of final states as $F = F_1 \times Q^b_2$. The idea behind this construction is that at the beginning the automaton simulates $\mathcal{A}_1$ and $\mathcal{A}_2$ with only $\circ$-transitions until $\mathcal{A}_1$ marks a position. At this point it goes on with the simulation of $\mathcal{A}_1$ and simultaneously verifies that the simulation of $\mathcal{A}_2$ can never pass through an accepting state, for which it uses the boolean automaton $\mathcal{A}^b_2$. Notice that we verify this in the construction when we consider only the transitions of Δ^b_2 that do not end in an accepting state.

Now, we prove the correctness of the above constructions. First, consider the `AND` case. Consider a match $M \in [\![\mathcal{A}]\!](S_i)$. Then, there is a run ρ of $\mathcal{A}$ of the form:

$$\rho : (q^1_0, q^2_0, \varnothing) \xrightarrow{\alpha_1/m_1} (q^1_1, q^2_1, Y_1) \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} (q^1_n, q^2_n, Y_n)$$

Because of the construction, each transition has the form $\alpha_i = \alpha^1_i \wedge \alpha^2_i$ such that the runs:

$$\rho_1 : q^1_0 \xrightarrow{\alpha^1_1/m_1} q^1_1 \xrightarrow{\alpha^1_2/m_2} \cdots \xrightarrow{\alpha^1_n/m_n} q^1_n$$

$$\rho_2 : q^2_0 \xrightarrow{\alpha^2_1/m_1} q^2_1 \xrightarrow{\alpha^2_2/m_2} \cdots \xrightarrow{\alpha^2_n/m_n} q^2_n$$

are accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, and $M = \mathrm{match}(\rho_1) = \mathrm{match}(\rho_2)$. By induction hypothesis there exist valuations ν_1 and ν_2 such that $M \in [\![\varphi_1]\!](S, i, \nu_1)$ and $M \in [\![\varphi_2]\!](S, i, \nu_2)$. Moreover, because of property (1), we know that every variable $x \in \mathrm{dom}(\nu_i)$ appears exactly once in ρ_i and only at the transition $\nu_i(x)$ of ρ_i. Because of the construction, no variable $x \in X$ can appear in two different transitions of ρ, which means that if it appears at some position of ρ_1, then it cannot appear at a different position of ρ_2, and conversely. Therefore, for every $x \in dom(\nu_1) \cap \mathrm{dom}(\nu_2)$ it holds that $\nu_1(x) = \nu_2(x)$. If we define $\nu = \nu_1[\nu_2 \to \mathrm{dom}(\nu_2)]$, then $M \in [\![\varphi_1]\!](S, i, \nu)$ and $M \in [\![\varphi_2]\!](S, i, \nu)$ still hold, thus $M \in [\![\varphi]\!](S, i, \nu)$. Moreover, by induction of property (1) over ρ_1 and ρ_2, and because of the construction, property (1) still holds for ν. For the opposite direction, consider $M \in [\![\varphi]\!](S, i, \nu)$ for some ν. By definition it means that $M \in [\![\varphi_1]\!](S, i, \nu)$ and $M \in [\![\varphi_2]\!](S, i, \nu)$ and, by induction hypothesis, there exist accepting runs:

$$\rho_1 : q_0^1 \xrightarrow{\alpha_1^1/m_1} q_1^1 \xrightarrow{\alpha_2^1/m_2} \cdots \xrightarrow{\alpha_n^1/m_n} q_n^1$$
$$\rho_2 : q_0^2 \xrightarrow{\alpha_1^2/m_1} q_1^2 \xrightarrow{\alpha_2^2/m_2} \cdots \xrightarrow{\alpha_n^2/m_n} q_n^2$$

Over $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that $\mathrm{match}(\rho_1) = \mathrm{match}(\rho_2) = M$. Moreover, because of property (2), we know that every $x \in \mathrm{vdef}_+(\varphi_i)$ that appears in a transition of ρ_i appears while reading $S[\nu(x)]$. Now, if we define the run:

$$\rho : (q_0^1, q_0^2, \varnothing) \xrightarrow{\alpha_1/m_1} (q_1^1, q_1^2, Y_1) \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} (q_n^1, q_n^2, Y_n)$$

Of $\mathcal{A}$, where each $\alpha_i = \alpha_i^1 \wedge \alpha_i^2$ and $Y_i = (\alpha_i) \cap X$, then clearly it is a valid run of $\mathcal{A}$, because for every two different transitions, say with α_j and α_k, it holds that $\alpha_j \cap \alpha_k \cap X = \varnothing$. Because $\mathrm{match}(\rho) = M$, then $M \in [\![\mathcal{A}]\!](S_i)$. Moreover, by induction of property (2) over ν with φ_1 and φ_2, and because of the construction, property (2) still holds for ρ.

Consider now the `ALL` case. Consider a match $M \in [\![\mathcal{A}]\!](S_i)$. Then, there is a run ρ of $\mathcal{A}$ of the form:

$$\rho : (q_0^1, q_0^2, \varnothing) \xrightarrow{\alpha_1/m_1} (q_1^1, q_1^2, Y_1) \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} (q_n^1, q_n^2, Y_n)$$

Where at some position j and some i, for all $k \geq j$ it is the case that $q_k^i = \bot$. Because of the construction, each transition has the form $\alpha_i = \alpha_i^1 \wedge \alpha_i^2$ such that the runs:

$$\rho_1 : q_0^1 \xrightarrow{\alpha_1^1/m_1^1} q_1^1 \xrightarrow{\alpha_2^1/m_2^1} \dots \xrightarrow{\alpha_{n_1}^1/m_{n_1}^1} q_{n_1}^1$$
$$\rho_2 : q_0^2 \xrightarrow{\alpha_1^2/m_1^2} q_1^2 \xrightarrow{\alpha_2^2/m_2^2} \dots \xrightarrow{\alpha_{n_2}^2/m_{n_2}^2} q_{n_2}^2$$

Are accepting runs of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, and $M = M_1 \cup M_2$, where $M_1 = \mathrm{match}(\rho_1)$ and $M_2 = \mathrm{match}(\rho_2)$, i.e., $m_j = \bullet$ only if $m_j^1 = \bullet$ or $m_j^2 = \bullet$. Notice that they do not have to end at the same time. By induction hypothesis, there exist valuations ν_1 and ν_2 such that $M \in [\![\varphi_1]\!](S, i, \nu_1)$ and $M \in [\![\varphi_2]\!](S, i, \nu_2)$. By the same reasoning of the `AND` case, for every $x \in dom(\nu_1) \cap \mathrm{dom}(\nu_2)$ it holds that $\nu_1(x) = \nu_2(x)$. If we define $\nu = \nu_1[\nu_2 \rightarrow \mathrm{dom}(\nu_2)]$, then $M_1 \in [\![\varphi_1]\!](S, i, \nu)$ and $M_2 \in [\![\varphi_2]\!](S, i, \nu)$ still hold, thus $M \in [\![\varphi]\!](S, i, \nu)$. Moreover, by induction of property (1) over ρ_1 and ρ_2, and because of the construction, property (1) still holds for ν. For the opposite direction, consider $M \in [\![\varphi]\!](S, i, \nu)$ for some ν. By definition it means that $M_1 \in [\![\varphi_1]\!](S, i, \nu)$ and $M_2 \in [\![\varphi_2]\!](S, i, \nu)$ for some M_1 and M_2 such that $M = M_1 \cup M_2$ and, by induction hypothesis, there exist accepting runs:

$$\rho_1 : q_0^1 \xrightarrow{\alpha_1^1/m_1^1} q_1^1 \xrightarrow{\alpha_2^1/m_2^1} \dots \xrightarrow{\alpha_{n_1}^1/m_{n_1}^1} q_{n_1}^1$$
$$\rho_2 : q_0^2 \xrightarrow{\alpha_1^2/m_1^2} q_1^2 \xrightarrow{\alpha_2^2/m_2^2} \dots \xrightarrow{\alpha_{n_2}^2/m_{n_2}^2} q_{n_2}^2$$

Over $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that $\mathrm{match}(\rho_1) = M_1$ and $\mathrm{match}(\rho_2) = M_2$. Moreover, because of property (2) and by the same reasoning of the `AND` case, then the run:

$$\rho : (q_0^1, q_0^2, \varnothing) \xrightarrow{\alpha_1/m_1} (q_1^1, q_1^2, Y_1) \xrightarrow{\alpha_2/m_2} \dots \xrightarrow{\alpha_n/m_n} (q_n^1, q_n^2, Y_n)$$

Where $m_j = \bullet$ only if $m_j^1 = \bullet$ or $m_j^2 = \bullet$, is a valid run of $\mathcal{A}$. Because $\mathrm{match}(\rho) = M_1 \cup M_2 = M$, then $M \in [\![\mathcal{A}]\!](S_i)$. Moreover, by induction of property (2) over ν with φ_1 and φ_2, and because of the construction, property (2) still holds for ρ.

Consider now the `UNLESS` case. Consider a match $M \in [\![\mathcal{A}]\!](S_i)$. Then, there is an accepting run ρ of $\mathcal{A}$ of the form:

$$\rho : (q_0, \mathbf{q_0}) \xrightarrow{\alpha_1/m_1} (q_1, \mathbf{q_1}) \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} (q_n, \mathbf{q_n})$$

Then, ρ can be spit at position $j = \min(M)$ as $\rho = \rho_1 \cdot \rho_2$ such that ρ_1 simulates runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ simultaneously with only $\circ$ transitions and ρ_2 simulates runs of $\mathcal{A}_1$ and $\mathcal{A}'_2$ simultaneously. Moreover, because of the construction, the run:

$$\rho_1 : q_0 \xrightarrow{\alpha_1/m_1} q_1 \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} q_n$$

Is an accepting run of $\mathcal{A}_1$ over S_i. Therefore, $\text{match}(\rho_1) = M \in [\![\mathcal{A}_1]\!](S_i)$, thus $M \in [\![\varphi_1]\!](S, i, \nu)$ for some ν. Now, by contradiction consider that there exist some M' and ν' such that $M' \in [\![\varphi_2]\!](S_i)$, with $\min(M) \leq \min(M')$ and $\max(M') \leq \max(M)$. Then, there exist the accepting runs:

$$\sigma^d : \mathbf{q_0} \xrightarrow{\beta_1^d/l_1} \mathbf{q_1} \xrightarrow{\beta_2^d/l_2} \cdots \xrightarrow{\beta_k^d/l_k} \mathbf{q_k}$$

$$\sigma^b : \mathbf{q_0} \xrightarrow{\beta_1^b} \mathbf{q_1} \xrightarrow{\beta_2^b} \cdots \xrightarrow{\beta_k^b} \mathbf{q_k}$$

Of $\mathcal{A}_2^d$ and $\mathcal{A}_2^b$, respectively such that $k \leq n$, $m'_i = \circ$ for all $i < j$ (recall that they are the only possible runs for that prefix of S because they are both deterministic). Then, because $\min(M) \leq \min(M')$ and $\max(M') \leq \max(M)$, for every run of $\mathcal{A}$ of the form:

$$\rho' : (q'_0, \mathbf{q_0}) \xrightarrow{\alpha'_1/m'_1} (q'_1, \mathbf{q_1}) \xrightarrow{\alpha'_2/m'_2} \cdots \xrightarrow{\alpha'_k/m'_k} (q'_k, \mathbf{q'_k})$$

Such that $m'_i = \circ$ for all $i < j$, it holds that $\mathbf{q'_k} \in F^b$. However, because of the construction there is no transition in Δ that gets to a state of the form $(p, \mathbf{q})$ with $\mathbf{q} \in F_b$, which is a contradiction. Therefore, there is no M' and ν' such that $M' \in [\![\varphi_2]\!](S_i)$, with $\min(M) \leq \min(M')$ and $\max(M') \leq \max(M)$, thus $M \in [\![\varphi]\!](S, i, \nu)$. The proof for the converse case follows directly from this one, it consists of following the steps in the opposite direction. Clearly, properties (1) and (2) still hold by induction hypothesis, only by keeping the same ν and ρ of the induction, respectively. $\square$

The previous result shows the cost that a CEP-system will have to incur if extended operators are used. It is important to note here that the quadratic or exponential cost is just with one extended operator, and this does not include the cost of bringing the formula into LP-normal form. Furthermore, this again shows the advantage of using safe formulas: if φ_1 AND φ_2 or φ_1 ALL φ_2 are safe, then the cost $2^{|X|}$ in their constructions can be avoided.

We now study how to build MA for CEPL formulas with selection strategies. For this, we present our results using a more general framework, in which selection strategies are applied directly over MA. Let $\mathcal{A}$ be a MA and SEL a selector in $\{\texttt{STRICT}, \texttt{NXT}, \texttt{MAX}\}$. Then we say that a MA $\mathcal{A}_{\texttt{SEL}}$ is equivalent to $\texttt{SEL}(\mathcal{A})$ whenever $[\![\texttt{SEL}(\mathcal{A})]\!](S) = [\![\mathcal{A}_{\texttt{SEL}}]\!](S)$ for every stream S.

We begin the discussion with the NXT-operator, probably the most interesting and meaningful selection strategy for CEP-systems. At first sight, one could think that it is impossible to build a MA for the NXT-semantics: a MA for computing the next semantics will have to remember and compare an unbounded number of matches which could lead to an unbounded amount of memory. Interestingly, this intuition is wrong since one can always build a MA for computing the next semantics and, moreover, this construction has a very useful property.

Theorem 7.3. *Let $\mathcal{A}$ be a MA with N states. Then, there is an N-ambiguous MA $\mathcal{A}_{\texttt{NXT}}$ that is equivalent to $\texttt{NXT}(\mathcal{A})$ and of size at most exponential in the size of $\mathcal{A}$.*

PROOF. Let $\mathcal{R}$ be a schema and $\mathcal{A} = (Q, \Delta, I, F)$ be a match automaton over $\mathcal{R}$. In order to define the new match automaton $\mathcal{A}_{\texttt{NXT}} = (Q_{\texttt{NXT}}, \Delta_{\texttt{NXT}}, I_{\texttt{NXT}}, F_{\texttt{NXT}})$ we first need to introduce some notation. We begin by imposing an arbitrary linear order $<$ between the states of Q, i.e., for every two different states $p, q \in Q$, either $p < q$ or $q < p$. Let $T_1 \dots T_k$ be a sequence of sets of states such that $T_i \subseteq Q$. We say that a sequence $T_1 \dots T_k$ is a *total preorder* over Q if $T_i \cap T_j = \varnothing$ for every $i \neq j$. Notice that the sequence is not necessarily a partition, i.e., it does not need to include all states of Q. A total preorder naturally defines a preorder between states where "p is less than q" whenever $p \in T_i$, $q \in T_j$, and $i < j$. For the sake of simplification, we define the concatenation between set

of states such that $T \cdot T' = TT'$ whenever T and T' are non-empty and $T \cdot T' = T \cup T'$ otherwise. The concatenation between sets will help to remove empty sets during the final construction. Now, given any sequence $T_1 \dots T_k$ (not necessarily a total preorder), one can convert $T_1 \dots T_k$ into a total preorder by applying the operation *Total Pre-Ordering* (TPO) defined as follows:

$$\mathrm{TPO}(T_1 \dots T_k) = U_1 \cdot \ldots \cdot U_k \quad \text{where } U_i = T_i - \bigcup_{j=1}^{i-1} T_j.$$

Let $F = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ be the set of all condition formulas in the transitions of Δ. Define the equivalence relation $=_F$ between tuples such that, for every pair of tuples t_1 and t_2, $t_1 =_F t_2$ holds if, and only if, both satisfy the same formulas, i.e., $t_1 \vDash \alpha_i$ holds iff $t_2 \vDash \alpha_i$ holds, for every i. Moreover, for every tuple t let $[t]_F$ represent the equivalence class of t defined by $=_F$, that is, $[t]_F = \{t' \mid t =_F t'\}$. Notice that, even though there are infinitely many tuples, there is a finite amount of equivalence classes which is bounded by all possible combinations of formulas in F, i.e., $2^{|F|}$. Now, for every t, define the formula:

$$\alpha_t = \Big(\bigwedge_{t \vDash \alpha_i} \alpha_i\Big) \wedge \Big(\bigwedge_{t \nvDash \alpha_i} \neg\alpha_i\Big)$$

and define the new set of formulas $F\text{-types} = \{\alpha_t \mid t \in \mathrm{tuples}(\mathcal{R})\}$. Notice that for every tuple t there is exactly one formula in $F\text{-types}$ that is satisfied by t, and that formula is precisely α_t. Finally, we extend the transition relation Δ as a function such that:

$$\Delta(T, \alpha, m) = \{q \in Q \mid \text{exist } p \in T \text{ and } \alpha' \in \mathbf{F}_u(\mathcal{R}) \text{ s.t. } \alpha \vDash \alpha' \text{ and } (p, (\alpha', m), q) \in \Delta\}$$

for every $T \subseteq Q$, $\alpha \in F\text{-types}$, and $m \in \{\bullet, \circ\}$.

In the sequel, we define the match automaton $\mathcal{A}_{\texttt{NXT}} = (Q_{\texttt{NXT}}, \Delta_{\texttt{NXT}}, I_{\texttt{NXT}}, F_{\texttt{NXT}})$ component by component. First, the set of states $Q_{\texttt{NXT}}$ is defined as follows:

$$Q_{\texttt{NXT}} = \{(T_1 \dots T_k, p) \mid T_1 \dots T_k \text{ is a total preorder over } Q \text{ and } p \in T_i \text{ for some } i \leq k\}$$

Intuitively, the state p is the current state of the 'simulation' of $\mathcal{A}$ and the sets $T_1 \dots T_k$ contain the states in which the automaton could be, considering the prefix of the word

read until the current moment. Furthermore, the sets are ordered consistently with respect to the next-match semantic, e.g., if a run ρ_1 reach the state $(\{1,2\}\{3\},1)$ and other run ρ_2 reach the state $(\{1,2\}\{3\},3)$, then $\mathrm{match}(\rho_2) < \mathrm{match}(\rho_1)$. This property is proven later in Lemma 7.1.

Secondly, the transition relation is defined as follows. Consider $\alpha \in F\text{-types}$, $m \in \{\bullet,\circ\}$ and $(\mathcal{T},p),(\mathcal{U},q) \in Q_{\texttt{NXT}}$ where $\mathcal{T} = T_1 \dots T_k$ and $p \in T_i$ for some $i \leq k$. Then we have that $((\mathcal{T},p),\alpha,m,(\mathcal{U},q)) \in \Delta_{\texttt{NXT}}$ if, and only if,

(i) $(p,\alpha',m,q) \in \Delta$ for some α' such that $\alpha \vDash \alpha'$,

(ii) $q \notin \Delta(T_j,\alpha,m')$ for every $m' \in \{\bullet,\circ\}$ and $j < i$,

(iii) $\mathcal{U} = \mathrm{TPO}(U_1^\bullet \cdot U_1^\circ \cdot \ldots \cdot U_k^\bullet \cdot U_k^\circ)$ where $U_j^\bullet = \Delta(T_j,\alpha,\bullet)$ and $U_j^\circ = \Delta(T_j,\alpha,\circ)$ for $1 \leq j \leq k$, and

(iv) $q \notin \Delta(T_i,\alpha,\bullet)$ when $m = \circ$,

(v) $(p',\alpha',m,q) \notin \Delta$ for every $p' \in T_i$ such that $p' < p$ and every α' such that $\alpha \vDash \alpha'$.

Intuitively, the first condition ensures that the 'simulation' respects the transitions of Δ, the second checks that the next state could not have been reached from a 'higher' run, the third ensures that the sequence is updated correctly and the fourth restricts that if the next state can be reached either marking the letter or not, it always choose to mark it. The last condition is not strictly necessary, and removing it will not change the semantics of the automaton, but is needed to ensure that $\mathcal{A}_{\texttt{NXT}}$ is finitely ambiguous. What it does is making sure that there are no two runs ρ_1 and ρ_2 that end in the same state such that $\mathrm{match}(\rho_1) = \mathrm{match}(\rho_2)$.

Finally, the initial set $I_{\texttt{NXT}}$ is defined as all states of the form (I,q) where $q \in I$ and the final set $F_{\texttt{NXT}}$ as all states of the form $(T_1 \dots T_k,p)$ such that $p \in F$ and there exists $i \leq k$ such that $p \in T_i$ and $T_j \cap F = \varnothing$ for all $j < i$.

Let $S = t_1 t_2 \dots$ be any stream. To prove that the construction is correct, we will need the following lemma.

Lemma 7.1. *Consider a MA $\mathcal{A} = (Q,\Delta,I,F)$, a stream S, two states $(\mathcal{T},p),(\mathcal{T},q) \in Q_{\texttt{NXT}}$ with the same sequence $\mathcal{T} = T_1 \dots T_k$ such that $p \in T_i$, $q \in T_j$ for some i and j, and*

two runs ρ_1, ρ_2 of $\mathcal{A}_{\texttt{NXT}}$ over S that have the same length and reach the states $(\mathcal{T}, p)$ and $(\mathcal{T}, q)$, respectively. Then, $i < j$ if, and only if:

$$\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$$

PROOF. We will prove it by induction over the length of the runs. Let $q_0, q_0' \in I$ be any two initial states of $\mathcal{A}$, not necessarily different. First, assume that both runs consist of reading a single tuple t. Then, the runs are of the form:

$$\rho_1 : (I, q_0) \xrightarrow{\alpha_t/m_1} (\mathcal{T}, p) \qquad \text{and} \qquad \rho_2 : (I, q_0') \xrightarrow{\alpha_t/m_2} (\mathcal{T}, q)$$

where $\mathcal{T} = T_1T_2 = \mathrm{TPO}(\Delta(I, \alpha_t, \bullet)\Delta(I, \alpha_t, \circ))$ and neither T_1 nor T_2 can be empty because p and q are in different sets. For the if direction, the only option is that $\mathrm{match}(\rho_1) = \{1\}$ and $\mathrm{match}(\rho_2) = \{\}$, which implies that $m_1 = \bullet$ and $m_2 = \circ$. Then $i < j$ because $p \in T_1$ and $q \in T_2$. For the only-if direction, because $i < j$ then $p \in T_1$ and $q \in T_2$, so necessarily $m_1 = \bullet$ and $m_2 = \circ$. Because of this, $\mathrm{match}(\rho_1) = \{1\}$ and $\mathrm{match}(\rho_1) = \{\}$, therefore $\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$. Now, let $S = t_1t_2 \ldots t_n \ldots$ and consider that the runs are of the form:

$$\rho_1 : (I, q_0) \xrightarrow{\alpha_{t_1}/m_1} (\mathcal{T}_1, q_1) \xrightarrow{\alpha_{t_2}/m_2} \ldots \xrightarrow{\alpha_{t_{n-1}}/m_{n-1}} (\mathcal{T}_{n-1}, q_{n-1}) \xrightarrow{\alpha_{t_n}/m_n} (\mathcal{T}, p)$$
$$\rho_2 : (I, q_0') \xrightarrow{\alpha_{t_1}/m_1'} (\mathcal{T}_1, q_1') \xrightarrow{\alpha_{t_2}/m_2'} \ldots \xrightarrow{\alpha_{t_{n-1}}/m_{n-1}'} (\mathcal{T}_{n-1}, q_{n-1}') \xrightarrow{\alpha_{t_n}/m_n'} (\mathcal{T}, q)$$

Notice that both runs have the same sequences $\mathcal{T}_1, \ldots, \mathcal{T}_{n-1}$ because each sequence $\mathcal{T}_i$ is defined only by the previous sequence $\mathcal{T}_{i-1}$ and the tuple t_i which implicitly defines the formula α_{t_i}. Furthermore, all the runs over the same word must have the same sequences. Define the runs ρ_1' and ρ_2', respectively, as the runs ρ_1 and ρ_2 without the last transition. Consider that $\mathcal{T}_{n-1}$ has the form $\mathcal{T}_{n-1} = U_1U_2 \ldots U_k$, and that $q_{n-1} \in U_r$ and $q_{n-1}' \in U_s$ for some r and s. Notice that, because of the construction, if it is the case that $r < s$ ($r > s$), then $i < j$ ($i > j$ resp.) must hold. For the if direction, consider that $\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$. If $\mathrm{match}(\rho_1') = \mathrm{match}(\rho_2')$, by induction hypothesis it means that $r = s$. Moreover, the only option is that $m_n = \bullet$ and $m_n' = \circ$, therefore, by the construction it holds that $i < j$. If $\mathrm{match}(\rho_2') \leq_{\mathrm{next}} \mathrm{match}(\rho_1')$, by induction hypothesis it means that $r < s$

and because of the construction, $i < j$. Notice that $\mathrm{match}(\rho_1') \leq_{\mathrm{next}} \mathrm{match}(\rho_2')$ cannot occur because the lower element of $\mathrm{match}(\rho_2')$ not in $\mathrm{match}(\rho_1')$ would still be the lower element of $\mathrm{match}(\rho_2)$ not in $\mathrm{match}(\rho_1)$, thus contradicting $\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$. For the only-if direction, consider that $i < j$. It is easy to see that, if $r > s$, then i cannot be lower than j, thus we do not consider this case. Now, consider the case that $r = s$. Because $i < j$, it must occur that $m_n = \bullet$ and $m_n' = \circ$, so $\mathrm{match}(\rho_1) = \mathrm{match}(\rho_1') \cup \{n\}$ and $\mathrm{match}(\rho_2) = \mathrm{match}(\rho_2')$. By induction hypothesis, $\mathrm{match}(\rho_1') = \mathrm{match}(\rho_2')$, therefore $\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$. Consider now the case that $r < s$. By induction hypothesis, $\mathrm{match}(\rho_2') \leq_{\mathrm{next}} \mathrm{match}(\rho_1')$ and, because the last transition can only add n to both matches, it follows that $\mathrm{match}(\rho_2) \leq_{\mathrm{next}} \mathrm{match}(\rho_1)$. □

Now, we need to prove that if $M \in [\![\mathtt{NXT}(\mathcal{A})]\!](S)$, then $M \in [\![\mathcal{A}_{\mathtt{NXT}}]\!](S)$ and vice versa. First, consider a match $M \in [\![\mathcal{A}_{\mathtt{NXT}}]\!](S)$. To prove that $M \in [\![\mathtt{NXT}(\mathcal{A})]\!](S)$, we need to show that $M \in [\![\mathcal{A}]\!](S)$ and that for all matches M' such that $M \leq_{\mathtt{NXT}} M'$ and $\max(M) = \max(M')$, $M' \notin [\![\mathcal{A}]\!](S)$. Assume that the run associated to M is:

$$\rho : (\mathcal{U}_0, q_0) \xrightarrow{\alpha_{t_1}/m_1} (\mathcal{U}_1, q_1) \xrightarrow{\alpha_{t_2}/m_2} \dots \xrightarrow{\alpha_{t_n}/m_n} (\mathcal{U}_n, q_n)$$

Because of the construction of the transition relation (in particular, the first condition), for every i it holds that $(q_{i-1}, \alpha_i, m_i, q_i) \in \Delta$ for some α_i such that $\alpha_{t_i} \vDash \alpha_i$. Because $t_i \vDash \alpha_{t_i}$, then $t_i \vDash \alpha_i$, thus the run:

$$\rho' : q_0 \xrightarrow{\alpha_1/m_1} q_1 \xrightarrow{\alpha_2/m_2} \dots \xrightarrow{\alpha_n/m_n} q_n$$

is an accepting run of $\mathcal{A}$ over S, and thus $M \in [\![\mathcal{A}]\!](S)$. Now, recall from construction of $F_{\mathtt{NXT}}$ that there exists $i \leq k$ such that $q_n \in T_i$ and $T_j \cap F = \varnothing$ for all $j < i$, where $T_1 \dots T_k = \mathcal{U}_n$. Then, because of Lemma 7.1, $M' \leq_{\mathrm{next}} M$ for every other $M' \in [\![\mathcal{A}]\!](w)$ such that $\max(M) = \max(M')$, otherwise the run of M' would end in a state inside a T_j such that $j < i$ which cannot happen. Therefore, $M \in [\![\mathtt{NXT}(\mathcal{A})]\!](S)$.

Now, consider a match $M \in [\![\mathtt{NXT}(\mathcal{A})]\!](S)$. Assume that the run associated to M is:

$$\rho : q_0 \xrightarrow{\alpha_1/m_1} q_1 \xrightarrow{\alpha_2/m_2} \dots \xrightarrow{\alpha_n/m_n} q_n$$

To prove that $M \in [\![\mathcal{A}_{\texttt{NXT}}]\!](S)$ we will prove that there exists an accepting run on $\mathcal{A}_{\texttt{NXT}}$. Based on ρ, consider now the run:

$$\rho' : (\mathcal{U}_0, p_0) \xrightarrow{\alpha_{t_1}/m_1} (\mathcal{U}_1, p_1) \xrightarrow{\alpha_{t_2}/m_2} \dots \xrightarrow{\alpha_{t_n}/m_n} (\mathcal{U}_n, p_n)$$

Where the matches $m_1, \dots, m_n$ are the same, each condition α_{t_i} is defined by t_i and each $\mathcal{U}_i$ is the result of applying the function TPO based on $\mathcal{U}_{i-1}$ and α_{t_i} Moreover, each p_i is defined as follows. As notation, consider that $\mathcal{U}_i = T^i_1 \dots T^i_{k_i}$ and that every q_i is in the r_i-th set of $\mathcal{U}_i$, i.e., $q_i \in T^i_{r_i}$. Then, p_i is the lower state in $T^i_{r_i}$ such that $(p_i, (\alpha_{t_{i+1}}, m_{i+1}, p_{i+1})\ in \Delta$, and $p_n = q_n$. Notice that ρ' is completely defined by ρ and S. We will prove that ρ' is an accepting run by checking that all transitions meet the conditions of the transition relation $\Delta_{\texttt{NXT}}$. Now, it is clear that the first condition is satisfied by all transitions, i.e., for every i it holds that $(p_{i-1}, \alpha', m_i, p_i) \in \Delta$ for some α' such that $\alpha_{t_i} \vDash \alpha'$ (just consider $\alpha' = \alpha_i$). For the second condition, by contradiction suppose that it is not satisfied by ρ'. It means that for some i, $p_i \in \Delta(T^{i-1}_j, a_i, m')$ for some $m' \in \{\bullet, \circ\}$ and $j < r_i$. In particular, consider that the state $p' \in T^{i-1}_j$ is the one for which $(p', a_i, m', q_i) \in \Delta$. Recall that every state inside a sequence is reachable considering the prefix of the word read until that moment. This means that there exist the accepting runs:

$$\sigma : q'_0 \xrightarrow{\alpha'_1/m'_1} q'_1 \xrightarrow{\alpha'_2/m'_2} \dots \xrightarrow{\alpha'_{i-1}/m'_{i-1}} q' \xrightarrow{\alpha'_i/m'_i} q_i \xrightarrow{\alpha_{i+1}/m_{i+1}} \dots \xrightarrow{\alpha_n/m_n} q_n$$

$$\sigma' : (\mathcal{U}_0, p'_0) \xrightarrow{\alpha_{t_1}/m'_1} (\mathcal{U}_1, p'_1) \xrightarrow{\alpha_{t_1}/m'_2} \dots \xrightarrow{\alpha_{t_i}/m'_i} (\mathcal{U}_i, p_i) \xrightarrow{\alpha_{t_{i+1}}/m_{i+1}} \dots \xrightarrow{\alpha_{t_n}/m_n} (\mathcal{U}_n, p_n)$$

Where p'_i are defined in a similar way to p_i. Define for every run γ and every i the run γ_i as γ until the i-th transition. For example, ρ_i is equal to the run ρ until the state q_i. Then, by Lemma 7.1, $\mathrm{match}(\rho'_{i-1}) < \mathrm{match}(\sigma'_{i-1})$, but $\mathrm{match}(\rho') = \mathrm{match}(\sigma')$. This is a contradiction, since $\mathrm{match}(\rho')$ and $\mathrm{match}(\sigma')$ differ from $\mathrm{match}(\rho'_{i-1})$ and $\mathrm{match}(\sigma'_{i-1})$ in that the latters can contain additional positions from i to n, but the minimum position remains in $\mathrm{match}(\sigma'_{i-1})$, and therefore in $\mathrm{match}(\sigma')$. The fourth condition is proven by contradiction too. Suppose that it is not satisfied by ρ', which means that for some i,

$p_i \in \Delta(T^{i-1}_{r_{i-1}}, \alpha_{t_i}, \bullet)$ when $m_i = \circ$. Then, the run:

$$\sigma : p_0 \xrightarrow{\alpha_{t_1}/m_1} p_1 \xrightarrow{\alpha_{t_2}/m_2} \dots \xrightarrow{\alpha_{t_{i-1}}/m_{i-1}} p_{i-1} \xrightarrow{\alpha_{t_i}/\bullet} p_i \xrightarrow{\alpha_{t_{i+1}}/m_{i+1}} \dots \xrightarrow{\alpha_{t_n}/m_n} p_n$$

is an accepting run such that $\mathrm{match}(\rho) < \mathrm{match}(\sigma)$, which is a contradiction, since $M \in [\![\mathtt{NXT}(\mathcal{A})]\!](S)$. The third and last conditions are trivially proven because of the construction of the run. Therefore, ρ' is a valid run of $\mathcal{A}_{\mathtt{NXT}}$ over S. Moreover, because $p_n = q_n \in F$ then ρ' is an accepting run, therefore $\mathrm{match}(\rho) = M \in [\![\mathcal{A}_{\mathtt{NXT}}]\!](S)$.

Now, we analyze the properties of the automaton $\mathcal{A}_{\mathtt{NXT}}$. First, we show that $|\mathcal{A}_{\mathtt{NXT}}|$ is at most exponential over $|\mathcal{A}|$. Notice that each state in $Q_{\mathtt{NXT}}$ represents a sequence of subsets of Q, thus each state has at most $|Q|$ subsets. Moreover, for each one of the subsets there are at most $2^{|Q|}$ possible combinations. Therefore, there are no more than $2^{|Q|}Q$ possible states in $Q_{\mathtt{NXT}}$, thus $|\mathcal{A}| \in \mathcal{O}(|\mathcal{A}|)$. Now, we know that for every accepting run:

$$\rho : (\mathcal{T}_0, q_0) \xrightarrow{\alpha_{t_1}/m_1} (\mathcal{T}_1, q_1) \xrightarrow{\alpha_{t_2}/m_2} \dots \xrightarrow{\alpha_{t_n}/m_n} (\mathcal{T}_n, q_n)$$

that ends in a position n, it holds that $M = \mathrm{match}(\rho)$ is the biggest among all matches M' such that $\max(M') = n$, according to $\leq_{\mathrm{next}}$. Moreover, because of Lemma 4.1 we know that there is only one maximum, meaning that any other accepting run:

$$\rho' : (\mathcal{T}_0, q'_0) \xrightarrow{\alpha_{t_1}/m'_1} (\mathcal{T}_1, q'_1) \xrightarrow{\alpha_{t_2}/m'_2} \dots \xrightarrow{\alpha_{t_n}/m'_n} (\mathcal{T}_n, q'_n)$$

That ends in position n must define the same match, i.e., $\mathrm{match}(\rho') = M$. Furthermore, for this to happen it must occur that both q_i and q'_i are in the same subset T^i_r of $\mathcal{T}_i$, for every $i \leq n$. Notice that if $\rho \neq \rho'$ then they cannot be both in the same state ate the same time, otherwise it would contradict the condition 5 in the construction of $\Delta_{\mathtt{NXT}}$. Because of this, the size of $\mathrm{Run}_n(\mathcal{A}, S)$ is bounded by the number of final states, i.e., $|\mathrm{Run}_n(\mathcal{A}, S)| \leq |F|$. Therefore $\mathcal{A}_{\mathtt{NXT}}$ is $|F|$-ambiguous. □

The previous result is very important for our framework, as one can always compute the next semantics of any MA and, moreover, the resulting MA is finitely ambiguous.

Combined with Theorem 6.1, this leads to an efficient evaluation strategy for unary CEPL formulas under the next semantics.

Corollary 7.1. *For every unary ecore-CEPL formula φ, there is an efficient evaluation strategy for* $\texttt{NXT}(\varphi)$. *Moreover, if $\mathcal{A}_\varphi$ is a MA equivalent to φ, the efficient evaluation strategy processes each event in time linear w.r.t. $\mathcal{A}_\varphi$.*

PROOF. In Algorithm 2 we provide an efficient strategy to evaluate a MA over the NXT semantics. Similar to the MA of the NXT construction, we need to keep track of the order of priority between alls runs. For this, the algorithm stores the runs in a queue structure E, which has the functions `enqueue`, to add a new element at the end, and `dequeue`, to extract the first element. Here, each element in E represents a tuple (T, M) where T is a non-empty set of states and M is a match. We use the function `notin` which receives a state q and a queue E, and has the value TRUE if q does not appear in any run of E, and FALSE otherwise.

For all (T, M) in E, every $q \in T$ represents a run of $\mathcal{A}$ whose associated partial match is precisely M. To this end, the subroutine `Update` computes the set T' of states that can be reached from all the states of T using a transition with mark m. Afterwards, it adds the new tuple (T', M') to E', where M' is equal to M if $m = \circ$, and is equal to M plus the position i currently read if $m = \bullet$. After applying `Update` over all the elements of E, the resulting set E' is similar to the result of the function TPO in Theorem 7.3, in the sense that the result only stores information about the runs that could lead to a match of the NXT semantics. The key to achieve this is that the higher runs are updated first (line 5) and, moreover, the update is first done using $\bullet$ transitions, and later using $\circ$ (lines 6 and 7). Therefore, at each iteration the queue $E = [(T_0, M_0), \ldots, (T_n, M_n)]$ is updated in such a way that:

- The matches follow the (reversed) $<_{\text{next}}$ order, i.e., $M_n <_{\text{next}} M_{n-1} <_{\text{next}} \ldots <_{\text{next}} M_0$.
- No state can appear twice in E, and

Algorithm 2 Evaluate $\mathcal{A} = (Q, \Delta, I, F)$ over a stream S with `NXT` semantics

```
1:  procedure EVAL[A](S)
2:      E ← [(I, ∅)]
3:      while t ← yield_S do
4:          E' ← [ ]
5:          for (T, M) ← E.dequeue() do
6:              E' ← Update(E', T, M, t, •)
7:              E' ← Update(E', T, M, t, ∘)
8:          end for
9:          E ← E'
10:         enumerate(arg min_M({ j | (T, M) = E[j] ∧ F ∩ T ≠ ∅ }))
11:     end while
12: end procedure
13: procedure Update[A](E', T, M, t, m)
14:     T' ← ∅
15:     for all q ∈ T ∧ (q, α, m, q') ∈ Δ do
16:         if t ⊨ α ∧ notin(q, E') then
17:             T' ← T' ∪ { q' }
18:         end if
19:     end for
20:     if T' ≠ ∅ then
21:         if m = • then
22:             E'.enqueue((T', M ∪ {i}))
23:         else
24:             E'.enqueue((T', M))
25:         end if
26:     end if
27:     return E'
28: end procedure
```

- For every $q \in T_i$, the match M_i is the highest of all the partial runs of $\mathcal{A}$ that could currently be in q.

Finally, after updating E the algorithm retrieves the highest match M (w.r.t. the $<_{\text{next}}$ order) if there exists a run associated to M currently in a final state q.

It is clear to see that Algorithm 2 iterates over all states of $\mathcal{A}$ (because each one appears at most one in E), and for each one it iterates over all transitions that begin at q. Therefore, if we assume that the transitions are indexed by their initial states, it is easy to see that the iteration for each event takes linear time over the size of $\mathcal{A}$. □

This result guarantees that, if the NXT-semantics is used over a unary CEPL formula, one can efficiently evaluate the formula over any stream. Furthermore, the size of $\mathcal{A}_{\texttt{NXT}(\varphi)}$ is important for the performance of the evaluation. For this reason, Theorems 7.1 and 7.2 must be considered to keep the construction as small as possible. For the final part of this section, we discuss how to compile the STRICT and MAX selection strategies. In the following result, we show that both operators can be run by MA. Unfortunately, they do not have the finitely ambiguous property as the NXT-operator.

Theorem 7.4. *For any MA $\mathcal{A}$, there is a MA $\mathcal{A}_{\texttt{STRICT}}$ equivalent to $\texttt{STRICT}(\mathcal{A})$. Further, $\mathcal{A}_{\texttt{STRICT}}$ is of size at most linear in the size of $\mathcal{A}$.*

PROOF. Consider a MA $\mathcal{A} = (Q, \Delta, I, F)$. We will first define a MA $\mathcal{A}_{\texttt{STRICT}} = (Q_{\texttt{STRICT}}, \Delta_{\texttt{STRICT}}, I_{\texttt{STRICT}}, F_{\texttt{STRICT}})$ and then prove that it is equivalent to $\texttt{STRICT}(\mathcal{A})$. The set of states is defined as $Q_{\texttt{STRICT}} = \{q^m \mid q \in Q \wedge m \in \{\bullet, \circ\}\}$, the transition relation is $\Delta_{\texttt{STRICT}} = \{(p^m, (\alpha, m), q^m) \mid (p, (\alpha, m), q) \in \Delta)\} \cup \{(p^\circ, (\alpha, \bullet), q^\bullet) \mid (p, (\alpha, \bullet), q) \in \Delta\}$, the initial states are $I_{\texttt{STRICT}} = \{q^\circ \mid q \in I\}$ and the final states are $F_{\texttt{STRICT}} = \{q^\bullet \mid q \in F\}$. Basically, there are two copies of $\mathcal{A}$, the first one which only have the $\circ$ transitions, and the second one which only have the $\bullet$ ones, and at any $\bullet$ transition it can move from the first on to the second. On an execution, $\mathcal{A}_{\texttt{STRICT}}$ starts in the first copy of $\mathcal{A}$, moving only through transitions that do not mark the positions, until it decides to mark one. At that point it moves to the second copy of $\mathcal{A}$, and from there on it moves only using transitions with $\bullet$ until it reaches an accepting state.

Now, we prove that the construction is correct, i.e., $[\![\mathcal{A}_{\texttt{STRICT}}]\!](S) = [\![\texttt{STRICT}(\mathcal{A})]\!](S)$ for every S. Let S be any stream. First, consider a match $M \in [\![\texttt{STRICT}(\mathcal{A})]\!](S)$. This means that $M \in [\![\mathcal{A}]\!](S)$ and that M has the form $M = \{m_0, m_1, \ldots, m_k\}$ with $m_i = m_{i-1} + 1$. Therefore, there is an accepting run of $\mathcal{A}$ of the form:

$$\rho : q_0 \xrightarrow{\alpha_1/\circ} q_1 \xrightarrow{\alpha_2/\circ} \cdots \xrightarrow{\alpha_{m_1-1}/\circ} q_{m_1-1} \xrightarrow{\alpha_{m_1}/\bullet} q_{m_1} \xrightarrow{\alpha_{m_2}/\bullet} \cdots \xrightarrow{\alpha_{m_k}/\bullet} q_{m_k}$$

Such that $\mathrm{match}(\rho) = M$. Consider now the run over $\mathcal{A}_{\texttt{STRICT}}$ of the form:

$$\rho' : q_0^\circ \xrightarrow{\alpha_1/\circ} q_1^\circ \xrightarrow{\alpha_2/\circ} \cdots \xrightarrow{\alpha_{m_1-1}/\circ} q_{m_1-1}^\circ \xrightarrow{\alpha_{m_1}/\bullet} q_{m_1}^\bullet \xrightarrow{\alpha_{m_2}/\bullet} \cdots \xrightarrow{\alpha_{m_k}/\bullet} q_{m_k}^\bullet$$

It is clear that all transitions of ρ' are in $\Delta_{\texttt{STRICT}}$, because the ones with $\circ$ are in the first copy of $\mathcal{A}$, the first one with $\bullet$ passes from the first copy to the second, and the following ones with $\bullet$ are in the second copy. Therefore ρ' is indeed run of $\mathcal{A}_{\texttt{STRICT}}$ over S, and because $q_{m_k} \in F$, then $q_{m_k}^\bullet \in F$ and ρ' is an accepting run. Moreover, $\mathrm{match}(\rho') = M$, thus $M \in [\![A_{\texttt{STRICT}}]\!](S)$.

Now, consider a match $M \in [\![A_{\texttt{STRICT}}]\!](S)$, of the form $M = \{m_0, m_1, \ldots, m_k\}$. It means that there is an accepting run of $\mathcal{A}_{\texttt{STRICT}}$ of the form:

$$\rho : q_0^\circ \xrightarrow{\alpha_1/\circ} \cdots \xrightarrow{\alpha_{m_1-1}/\circ} q_{m_1-1}^\circ \xrightarrow{\alpha_{m_1}/\bullet} q_{m_1}^\bullet \xrightarrow{\alpha_{m_2}/\bullet} \cdots \xrightarrow{\alpha_{m_k}/\bullet} q_{m_k}^\bullet$$

Such that $\mathrm{match}(\rho) = M$. Notice that ρ must have this form because of the structure of $\mathcal{A}_{\texttt{STRICT}}$, which force ρ to have $\circ$ transitions at the beginning and $\bullet$ ones at the end. Consider then the run of $\mathcal{A}$ of the form:

$$\rho' : q_0 \xrightarrow{\alpha_1/\circ} \cdots \xrightarrow{\alpha_{m_1-1}/\circ} q_{m_1-1} \xrightarrow{\alpha_{m_1}/\bullet} q_{m_1} \xrightarrow{\alpha_{m_2}/\bullet} \cdots \xrightarrow{\alpha_{m_k}/\bullet} q_{m_k}$$

Similar to the converse case, it is clear that all transitions in ρ' are in Δ. Therefore ρ' is an accepting run of $\mathcal{A}$ over S, and because $\mathrm{match}(\rho') = M$, it holds that $M \in [\![\texttt{STRICT}(\mathcal{A})]\!](S)$.

Finally, notice that $\mathcal{A}_{\texttt{STRICT}}$ consists in duplicating $\mathcal{A}$, thus the size of $\mathcal{A}_{\texttt{STRICT}}$ is two times the size of $\mathcal{A}$. □

Theorem 7.5. *For any MA $\mathcal{A}$, there is a MA $\mathcal{A}_{\texttt{MAX}}$ equivalent to $\texttt{MAX}(\mathcal{A})$. Further, $\mathcal{A}_{\texttt{MAX}}$ is of size at most double exponential in the size of $\mathcal{A}$.*

PROOF. Let $\mathcal{A} = (Q, \Delta, q_0, F)$ be a match automaton. Without lost of generality, we assume that $\mathcal{A}$ is deterministic. If not, one can determinize $\mathcal{A}$ incurring in an extra exponential blow-up in the number of states. Similarly to the construction of MA for the NXT, we define the set F-types such that for every tuple t there is exactly one formula

α_t in F-types that is satisfied by t, and extend the transition relation Δ as a function $\Delta(T, \alpha, m)$ for every $T \subseteq Q$, $\alpha \in F$-types, and $m \in \{\bullet, \circ\}$. Similarly, we overload the notation of Δ as a function such that $\Delta(T, \alpha) = \Delta(T, \alpha, \bullet) \cup \Delta(T, \alpha, \circ)$.

We define the match automaton $\mathcal{A}_{\texttt{MAX}} = (Q_{\texttt{MAX}}, \Delta_{\texttt{MAX}}, I_{\texttt{MAX}}, F_{\texttt{MAX}})$ such that $Q_{\texttt{MAX}} = Q \times 2^Q$, $I_{\texttt{MAX}} = \{(q_0, \varnothing)\}$, and $F_{\texttt{MAX}} = \{(q, T) \in Q_{\texttt{MAX}} \mid q \in F \text{ and } T \cap F = \varnothing\}$. For the transition relation $\Delta_{\texttt{MAX}}$ we distinguish two cases depending on whether the transition is marking or not. For the unmarking transition we have that $((p, T), (\alpha, \circ), (q, U)) \in \Delta_{\texttt{MAX}}$ iff $U = \Delta(T, \alpha) \cup \Delta(\{p\}, \alpha, \bullet)$, $q \notin U$ and there is a formula $\alpha' \in \mathbf{F}_u(\mathcal{R})$ such that $(p, (\alpha', \circ), q) \in \Delta$ and $\alpha \vDash \alpha'$, for every $(p, T), (q, U) \in Q_{\texttt{MAX}}$ and $\alpha \in F$-types. On the other hand, for the marking transition we have that $((p, T), (\alpha, \bullet), (q, U)) \in \Delta_{\texttt{MAX}}$ iff $U = \Delta(T, \alpha, \bullet)$, $q \notin U$ and there is a formula $\alpha' \in \mathbf{F}_u(\mathcal{R})$ such that $(p, (\alpha', \bullet), q) \in \Delta$ and $\alpha \vDash \alpha'$, for every $(p, T), (q, U) \in Q_{\texttt{MAX}}$ and $\alpha \in F$-types.

Next, we prove the above, i.e., $M \in [\![\texttt{MAX}(\mathcal{A})]\!](S)$ iff $M \in [\![\mathcal{A}_{\texttt{MAX}}]\!](S)$. First, we prove the if direction. Consider a match M such that $M \in [\![\mathcal{A}_{\texttt{MAX}}]\!](S)$. To prove that $M \in [\![\texttt{MAX}(\mathcal{A})]\!](S)$, we first prove that $M \in [\![\mathcal{A}]\!](S)$ by giving an accepting run of $\mathcal{A}$ associated to M. Assume that the run of $\mathcal{A}_{\texttt{MAX}}$ over S associated to M is:

$$\rho : (q_0, T_0) \xrightarrow{\alpha_{t_1}/m_1} (q_1, T_1) \xrightarrow{\alpha_{t_2}/m_2} \dots \xrightarrow{\alpha_{t_n}/m_n} (q_n, T_n)$$

Where $T_0 = \varnothing$, $T_n \cap F = \varnothing$ and $((q_{i-1}, T_{i-1}), (\alpha_{t_i}, m_i), (q_i, T_i)) \in \Delta_{\texttt{MAX}}$. Furthermore, $q_0 \in I$ and $q_n \in F$. Also, from the construction of $\Delta_{\texttt{MAX}}$, we deduce that for every i there is a formula α_i such that $(q_{i-1}, (\alpha_i, m_i), q_i) \in \Delta$. This means that the run:

$$q_0 \xrightarrow{\alpha_1/m_1} q_1 \xrightarrow{\alpha_2/m_2} \dots \xrightarrow{\alpha_n/m_n} q_n$$

Is an accepting run of $\mathcal{A}$ associated to M. Now, we prove by contradiction that for every M' such that $M \sqsubset M'$, $M' \notin [\![\mathcal{A}]\!](S)$. In order to do this, we define the next lemma, in which we use the notion of *partial run*, which is the same as a run but not necessarily beginning at an initial state.

Lemma 7.2. *Consider a deterministic match automaton $\mathcal{A} = (Q, \Delta, I, F)$, a stream $S = t_1, t_2, \ldots$, a partial run $\sigma : (q_0, T_0) \xrightarrow{\alpha_{t_1}/m_1} (q_1, T_1) \xrightarrow{\alpha_{t_2}/m_2} \cdots \xrightarrow{\alpha_{t_n}/m_n} (q_n, T_n)$ of $\mathcal{A}_{\texttt{MAX}}$ over S and a partial run $\sigma' : p_0 \xrightarrow{\alpha_1/m'_1} p_1 \xrightarrow{\alpha_2/m'_2} \cdots \xrightarrow{\alpha_n/m'_n} p_n$ of $\mathcal{A}$ over S. Then, if $p_0 \in T_0$ and $m'_i = \bullet$ at every i for which $m_i = \bullet$, it holds that $p_n \in T_n$.*

PROOF. This is proved by induction over the length n. First, if $n = 0$, then $p_n = p_0$ and $T_n = T_0$, so $p_n \in T_n$. Now, assume that the lemma holds for $n - 1$, i.e., $p_{n-1} \in T_{n-1}$. Consider the case that $m_n = \bullet$. Then $m'_n = \bullet$ too, thus $(p_{n-1}, (\alpha_n, \bullet), p_n) \in \Delta$. Furthermore, $T_n = \Delta(T_{n-1}, \alpha_{t_n}, \bullet)$ and therefore $p_n \in T_n$, because $p_{n-1} \in T_{n-1}$. Now, consider the case $m_n = \circ$. Either $(p_{n-1}, (\alpha_n, \bullet), p_n) \in \Delta$ or $(p_{n-1}, (\alpha_n, \circ), p_n) \in \Delta$, so $p_n \in \Delta(T_{n-1}, \alpha_{t_n})$. Moreover, $\Delta(T_{n-1}, \alpha_{t_n}) \subseteq T_n$ because of the construction of $\Delta_{\texttt{MAX}}$, therefore $p_n \in T_n$. □

Now, by contradiction consider a match M' such that $M \subset M'$ and $M' \in [\![\mathcal{A}]\!](S)$. Then, there must exist an accepting run of $\mathcal{A}$ over S associated to M' of the form:

$$\rho' : p_0 \xrightarrow{\alpha'_1/m'_1} p_1 \xrightarrow{\alpha'_2/m'_2} \cdots \xrightarrow{\alpha'_n/m'_n} p_n$$

such that $m'_i = \bullet$ at every i for which $m_i = \bullet$, and there is at least one i for which $m_i = \circ$ and $m'_i = \bullet$. Consider i to be the lower position for which this happens. Because $\mathcal{A}$ is deterministic, ρ' can be rewritten as:

$$\rho' : q_0 \xrightarrow{\alpha_1/m_1} \cdots \xrightarrow{\alpha_{i-1}/m_{i-1}} q_{i-1} \xrightarrow{\alpha'_i/\bullet} p_i \xrightarrow{\alpha'_{i+1}/m'_{i+1}} \cdots \xrightarrow{\alpha'_n/m'_n} p_n$$

Similarly, to ease visualization we rewrite ρ as:

$$\rho : (q_0, T_0) \xrightarrow{\alpha_{t_1}/m_1} \cdots \xrightarrow{\alpha_{t_{i-1}}/m_{i-1}} (q_{i-1}, T_{i-1}) \xrightarrow{\alpha_{t_i}/\circ} (q_i, T_i) \xrightarrow{\alpha_{t_{i+1}}/m_{i+1}} \cdots \xrightarrow{\alpha_{t_n}/m_n} (q_n, T_n)$$

In particular, the transition $((q_{i-1}, T_{i-1}), (\alpha_{t_i}, \circ), (q_i, T_i))$ is in $\Delta_{\texttt{MAX}}$, which means that $\Delta(\{q_{i-1}\}, \alpha_{t_i}, \bullet) \subseteq T_i$. Moreover, $(q_{i-1}, (\alpha'_i, \bullet), p_i) \in \Delta$ and, because $t_i \vDash \alpha'_i$, then $\alpha_{t_i} \vDash \alpha'_i$ thus $p_i \in T_i$. Now, by Lemma 7.2 it follows that $p_n \in T_n$. But, because ρ is an accepting run, $T_n \cap F = \varnothing$ and so $p_n \notin F$, which is a contradiction to the statement that ρ' is an accepting run. Therefore, for every M' such that $M \subset M'$, $M' \notin [\![\mathcal{A}]\!](S)$, hence $M \in [\![\texttt{MAX}(\mathcal{A})]\!](S)$.

Next, we will prove the only-if direction. For this, we will need the following lemma:

Lemma 7.3. *Consider a deterministic match automaton $\mathcal{A} = (Q, \Delta, I, F)$, a stream $S = t_1, t_2, \ldots$, a run $\sigma : (q_0, T_0) \xrightarrow{\alpha_{t_1}/m_1} (q_1, T_1) \xrightarrow{\alpha_{t_2}/m_2} \cdots \xrightarrow{\alpha_{n_2}/m_n} (q_n, T_n)$ of $\mathcal{A}_{\texttt{MAX}}$ over S and a state $p \in Q$. If $p \in T_n$. Then there is a run $\sigma' : p_0 \xrightarrow{\alpha_1/m'_1} p_1 \xrightarrow{\alpha_2/m'_2} \cdots \xrightarrow{\alpha_{n-1}/m'_{n-1}} p_{n-1} \xrightarrow{\alpha_n/m'_n} p$ of $\mathcal{A}$ over S such that* $\mathrm{match}(\sigma) \subset \mathrm{match}(\sigma')$.

PROOF. It will be proved by induction over the length n. The base case is $n = 0$, which is trivially true because $T_0 = \varnothing$. Assume now that the Lemma holds for $n - 1$. Define the run σ_{n-1} as the run σ without the last transition. For any state $q \in T_{n-1}$, let σ'_q be the run that ends in q such that $\mathrm{match}(\sigma_{n-1}) \subset \mathrm{match}(\sigma'_q)$. Consider the case $m_n = \circ$. Then, either $p \in \Delta(T_{n-1}, \alpha_{t_n})$ or $p \in \Delta(\{q_{n-1}\}, \alpha_{t_n}, \bullet)$. In the former scenario, there must be a $q \in T_{n-1}$ and $\alpha \in \mathbf{F}_u(\mathcal{R})$ such that $(q, (\alpha_n, m), p) \in \Delta$ and $\alpha_{t_n} \vDash \alpha$, with $m \in \{\bullet, \circ\}$. Define σ' as the run σ'_q followed by the transition $(q, (\alpha, m), p)$. Then σ' satisfies $\mathrm{match}(\sigma) \subset \mathrm{match}(\sigma')$. In the latter scenario, there must be an $\alpha \in \mathbf{F}_u(\mathcal{R})$ such that $(q_{n-1}, (\alpha, \bullet), p) \in \Delta$ and $\alpha_{t_n} \vDash \alpha$. Define σ' as σ_{n-1} followed by the transition $(q_{n-1}, (\alpha, \bullet), p)$. Then σ' satisfies $\mathrm{match}(\sigma) \subset \mathrm{match}(\sigma')$. Now, consider the case $m_n = \bullet$. Here, p has to be in $\Delta(T_{n-1}, \alpha_{t_n}, \bullet)$, so there must be a $q \in T_{n-1}$ and $\alpha \in \mathbf{F}_u(\mathcal{R})$ such that $(q, (\alpha, \bullet), p) \in \Delta$ and $\alpha_{t_n} \vDash \alpha$. Define σ' as the run σ'_q followed by the transition $(q, (\alpha, \bullet), p)$. Then σ' satisfies $\mathrm{match}(\sigma) \subset \mathrm{match}(\sigma')$. Finally, the Lemma holds for every n. □

Consider a match M such that $M \in [\![\texttt{MAX}(\mathcal{A})]\!](S)$. This means that there is an accepting run of $\mathcal{A}$ over S associated to M. Define that run as:

$$\rho : q_0 \xrightarrow{\alpha_1/m_1} q_1 \xrightarrow{\alpha_2/m_2} \cdots \xrightarrow{\alpha_n/m_n} q_n$$

Where $q_0 \in I$, $q_n \in F$ and $(q_{i-1}, (\alpha_i, m_i), q_i) \in \Delta$. To prove that $M \in [\![\mathcal{A}_{\texttt{MAX}}]\!](S)$ we give an accepting run of $\mathcal{A}_{\texttt{MAX}}$ over S associated to M. Consider the run:

$$\rho' : (q_0, T_0) \xrightarrow{\alpha_{t_1}/m_1} (q_1, T_1) \xrightarrow{\alpha_{t_2}/m_2} \cdots \xrightarrow{\alpha_{t_n}/m_n} (q_n, T_n)$$

Where $T_0 = \varnothing$, $T_i = \Delta(T_{i-1}, \alpha_{t_i}) \cup \Delta(\{q_{i-1}\}, \alpha_{t_i}, \bullet)$ if $m_i = \circ$, and $T_i = \Delta(T_{i-1}, \alpha_{t_i}, \bullet)$ if $m_i = \bullet$. To be a valid run, every transition $(T_{i-1}, \alpha_{t_i}, m_i, T_i)$ must be in $\Delta_{\texttt{MAX}}$, which we prove now by induction over i. The base case is $i = 0$, which is trivially true because no transition is required to exist. Next, assume that transitions up to $i-1$ exist. We know that there is an α_i such that $\alpha_{t_i} \vDash \alpha_i$ and $(q_{i-1}, (\alpha_i, m_i), q_i) \in \Delta$, so that condition is satisfied. We only need to prove that $q_i \notin T_i$. By contradiction, assume that $q_i \in T_i$. Consider the case that $m_i = \circ$. It means that either $q_i \in \Delta(\{q_{i-1}\}, \alpha_{t_i}, \bullet)$ or $q_i \in \Delta(T_{i-1}, \alpha_{t_i})$. In the first scenario, consider a new run σ to be exactly the same as ρ, but changing m_i with $\bullet$. Then σ is also an accepting run, and $\text{match}(\rho) \subset \text{match}(\sigma)$, which is a contradiction to the definition of the maximal semantic. In the second scenario, there must be some $p \in T_{i-1}$ and $\alpha \in \mathbf{F}_u(\mathcal{R})$ such that $(p, (\alpha, m), q_i) \in \Delta$, where $m \in \{\bullet, \circ\}$. Because of Lemma 7.3, it means that there is a run σ' over S:

$$\sigma' : p_0 \xrightarrow{\alpha'_1/m'_1} p_1 \xrightarrow{\alpha'_2/m'_2} \dots \xrightarrow{\alpha'_{i-2}/m'_{i-2}} p_{i-2} \xrightarrow{\alpha'_{i-1}/m'_{i-1}} p$$

Such that $\text{match}(\rho_{i-1}) \subset \text{match}(\sigma')$, where ρ_{i-1} is the run ρ until transition $i-1$. Moreover, because $(p, (\alpha, m), q_i) \in \Delta$ we can define the run:

$$\sigma : p_0 \xrightarrow{\alpha'_1/m'_1} p_1 \xrightarrow{\alpha'_2/m'_2} \dots \xrightarrow{\alpha'_{i-1}/m'_{i-1}} p \xrightarrow{\alpha/m} q_i \xrightarrow{\alpha_{i+1}/m_{i+1}} \dots \xrightarrow{\alpha_n/m_n} q_n$$

Such that $\text{match}(\rho) \subset \text{match}(\sigma)$, which is also a contradiction. Then, $q_i \notin T_i$ for the case $m_i = \circ$. Now, consider the case $m_i = \bullet$. Assuming that $q_i \in T_i$, it means that $q_i \in \Delta(T_{i-1}, \alpha_{t_i}, \bullet)$. Then, there must be some $p \in T_{i-1}$ and $\alpha \in \mathbf{F}(\mathcal{R})$ such that $(p, (\alpha, \bullet), q_i) \in \Delta$. Alike the previous case, because of Lemma 7.3, there is a run:

$$\sigma : p_0 \xrightarrow{\alpha'_1/m'_1} p_1 \xrightarrow{\alpha'_2/m'_2} \dots \xrightarrow{\alpha'_{i-1}/m'_{i-1}} p \xrightarrow{\alpha/\bullet} q_i \xrightarrow{\alpha_{i+1}/m_{i+1}} \dots \xrightarrow{\alpha_n/m_n} q_n$$

Such that $\text{match}(\rho) \subset \text{match}(\sigma)$, which is a contradiction. Then, $q_i \notin T_i$, therefore $(T_{i-1}, (\alpha_{t_i}, m_i), T_i) \in \Delta_{\texttt{MAX}}$ for every i. The above proved that ρ' is a run of $\mathcal{A}_{\texttt{MAX}}$, but to be a accepting run it must hold that $T_n \cap F = \varnothing$. By contradiction, assume otherwise, i.e., there is some $q \in Q$ such that $q \in T_n \cup F$. Then, because of Lemma 7.3, there is another

accepting run σ of $\mathcal{A}_{\texttt{MAX}}$ over S such that $M \subset \mathrm{match}(\sigma)$, which contradicts the fact that M is maximal. Thus, $T_n \cap F = \varnothing$ and ρ' is an accepting run, therefore $M \in [\![\mathcal{A}_{\texttt{MAX}}]\!](S)$.

It is clear that $\mathcal{A}_{\texttt{MAX}}$ is of size exponential in the size of $\mathcal{A}$ if this is deterministic, and double exponential if not (because of the exponential cost of determinizing it). □

Probably, the most unintuitive result above is that the MAX-operator can also be evaluated with MA. Similar to the NXT-operator, it seems hard to believe that there exists a MA that can keep an unbounded number of different maximal matches with a finite number of states. However, this can be done with finite memory but with a double exponential blow-up in the number of states. This shows that MA is a powerful model to evaluate CEPL formulas, and can be further exploited to evaluate selections strategies. Finally, the main disadvantage of these two operators is that the result is not necessarily finitely ambiguous and, therefore, we do not know how to evaluate $\mathcal{A}_{\texttt{STRICT}}$ and $\mathcal{A}_{\texttt{MAX}}$ efficiently.

8. EVALUATION OF UNARY CEPL

In this section, we put all pieces together and present a framework for efficiently evaluating CEPL formulas. In light of the results of Section 7, we evaluate unary formulas of the form $\texttt{NXT}(\varphi)$, for which we know there is an efficient evaluation strategy (Corollary 7.1). Although this is a fragment of CEPL, it settles the bases for a more general framework.

In Figure 8.1, we show the evaluation cycle of a CEPL formula in our framework with the main modules and partial results. For understanding the evaluation cycle, consider an input formula of the form $\texttt{NXT}(\varphi)$ where φ is a unary ecore-CEPL formula. The processing of $\texttt{NXT}(\varphi)$ starts in the Parser module, where we check if φ is well-formed (WF) and safe. These conditions are important to ensure that $\texttt{NXT}(\varphi)$ is streamable (Theorem 5.1) and satisfiable. Although unsafe formulas are not necessarily unsatisfiable, if a CEP system wants to allow unsafe formulas, it will have to assume an exponential blow-up in rewriting φ into its safe version (Theorem 5.2). Therefore, for complexity reasons our framework only receives safe formulas without losing expressive power whenever φ is a core-CEPL formula (recall Theorem 5.2).

The next module (Query rewrite) rewrites a well-formed and safe formula φ into a formula φ' in LP-normal form. For transforming CEPL formulas into LP-normal form, one can use the rewriting process of Theorem 5.3 which, in the worst case, can produce an exponential blow-up in the size of φ'. To avoid this cost, in many cases one can apply *local rewriting rules* which has been extensively studied in relational database management systems (DBMS) (Abiteboul et al., 1995; Ramakrishnan & Gehrke, 2003). For example, formula φ_1 in Section 2 is converted into φ_1' by applying a *filter push* on $(x.tmp > 40 \wedge x.id = 0)$ and $(y.hum <= 25 \wedge y.id = 0)$, avoiding the exponential blow-up of Theorem 5.3. As in DBMS, this approach can produce formulas in LP-normal form of polynomial size (w.r.t. φ). Unfortunately, we cannot apply this technique over formulas like φ_6 in Section 5, maintaining the blow-up of Theorem 5.3. Despite this, formulas like

φ_6 are rather uncommon in practice, and therefore we can assume that local rewriting rules will usually produce LP-formulas of polynomial size.

The third component, Compilation, receives formula φ' in LP-normal form and builds a match automaton $\mathcal{A}_\varphi$. For representing match automata we can use a compact data structure (e.g. a transition table (Aho, 1990; Cox, 2007)). By Theorem 7.1, we can construct $\mathcal{A}_\varphi$ in polynomial time in the size of φ' whenever φ is a core-CEPL formula. On the contrary, if φ is an ecore-CEPL formula, one has to afford an exponential blow-up with respect to the number of `AND`, `ALL` and `UNLESS` in φ (Theorem 7.2). As the number of extended operators is rather low in practice, however, the cost of compiling ecore-CEPL formulas should not affect the overall performance.

The last module (Evaluation) takes the MA $\mathcal{A}_\varphi$ produced by the Compilation module and evaluates it by using the `NXT` selector. For this, we can exploit the efficient evaluation strategy of Corollary 7.1. Moreover, this uses constant time per event (polynomial w.r.t. $\mathcal{A}_\varphi$). Note that the use of `NXT` is crucial for evaluating φ over the input stream efficiently: from our current results it is not clear whether the same holds for `STRICT` or `MAX` selection strategies.

Summing up, our framework can process queries of the form $\texttt{NXT}(\varphi)$ efficiently with time and space per item proportional to φ if the Query rewrite and Compilation module do not increase the size of φ' and $\mathcal{A}_\varphi$ significantly. Given that φ is small with respect to the (unbounded) input stream, one can assume constant time and space processing per-item (i.e. without considering the size of the output). It is important to note that the design of our framework highlights the modules that can increase the evaluation cost. As it was argued above, formulas that are difficult to rewrite into LP-normal form or contain too many extended operators should not be very common in practice. Still, one can consider new techniques, models, and selection strategies for the evaluation cycle in our framework that could overcome these pitfalls.

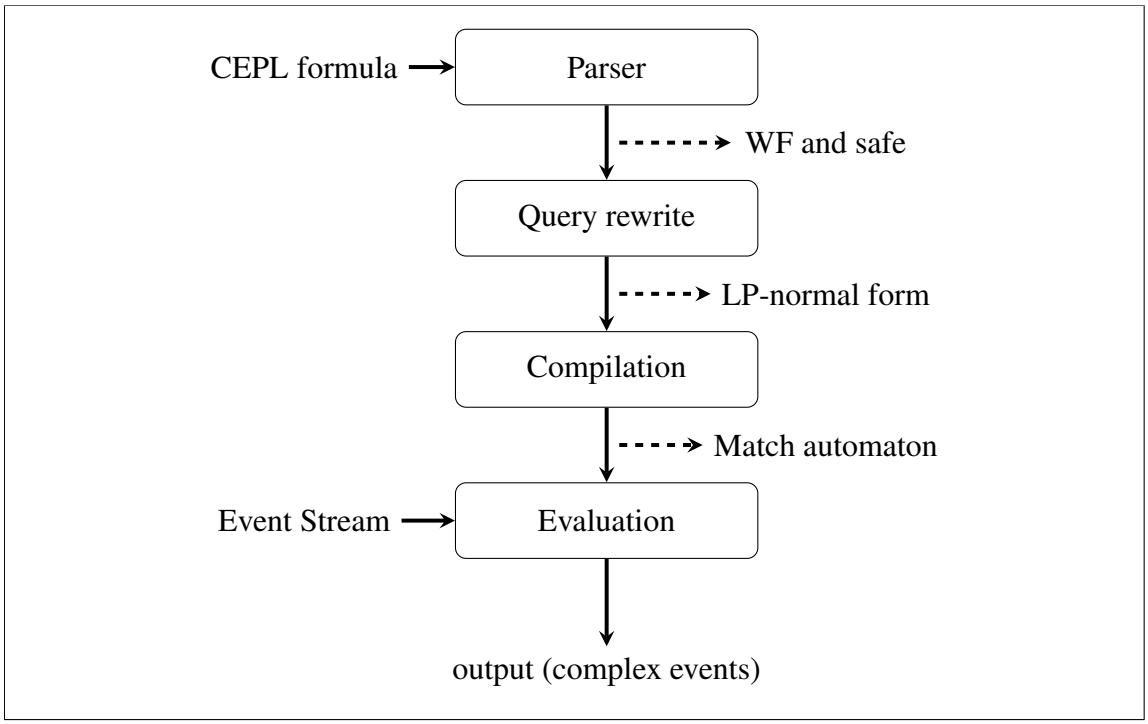

FIGURE 8.1. Evaluation framework for CEPL.

9. CONCLUSIONS AND FUTURE WORK

We have presented a formal framework for Complex Event Processing. We studied and formalized the different operators found in the literature and the so-called *selection strategies*, and we introduced a logic called CEPL that captures the main features of CEP. Towards building a framework for evaluating this language, we provided many interesting concepts and results for CEPL like syntactic restrictions (well-formed and safe), the LP-normal form, an evaluation model (MA), translation from unary CEPL to MA, and efficient evaluation of MA with finite ambiguity, among others. By gathering all these results together, we proposed the first formal framework for efficiently evaluating unary CEPL.

This thesis settles the basic foundations for CEP, stimulating many further research directions. In particular, a natural next step is the study of the evaluation of non-unary CEPL formulas, which require new insight in the rewriting of formulas and new computational models. Furthermore, a relevant problem for the area is to provide efficient evaluation strategies for these new computational models, or even for the full class of MA. Another problem in this line is the design of new selection strategies. In Section 4 we introduce three important selection strategies but one can envision many other useful strategies that could boost the evaluation of queries.

Finally, we have studied the fundamental features of CEP languages, leaving other features outside in order to keep the language and analysis simple. These features include time windows, aggregation, consumption policies, among others (see (Cugola & Margara, 2012b) for a more exhaustive list). We believe that each of these features can be used to extend CEPL in new directions to establish more complete frameworks for CEP.

REFERENCES

Abadi, D., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Erwin, C., ... Zdonik, S. (2003). Aurora: A data stream management system. In *Sigmod.*

Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases: the logical level*. Addison-Wesley.

Adi, A., & Etzion, O. (2004). Amit-the situation manager. *VLDB Journal*.

Agrawal, J., Diao, Y., Gyllstrom, D., & Immerman, N. (2008). Efficient pattern matching over event streams. In *Sigmod.*

Aho, A. V. (1990). Algorithms for finding patterns in strings. In *Handbook of theoretical computer science.*

Akdere, M., Çetintemel, U., & Tatbul, N. (2008). Plan-based complex event detection across distributed sources. *Proceedings of the VLDB Endowment*.

Alur, R., & Dill, D. L. (1994). A theory of timed automata. *Theoretical computer science*.

Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., & Studer, R. (2010). A rule-based language for complex event processing and reasoning. In *Rr.*

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., ... Widom, J. (2003). Stream: The stanford stream data manager (demonstration description). In *Sigmod.*

Arasu, A., Babu, S., & Widom, J. (2006). The cql continuous query language: Semantic foundations and query execution. *The VLDB Journal*.

Bagan, G., Durand, A., & Grandjean, E. (2007). On acyclic conjunctive queries and constant delay enumeration. In *Proc. csl.*

Barga, R. S., Goldstein, J., Ali, M. H., & Hong, M. (2007). Consistent streaming through time: A vision for event stream processing. In *Cidr.*

Berstel, J. (2013). *Transductions and context-free languages*. Springer-Verlag.

Buchmann, A., & Koldehofe, B. (2009). Complex event processing. *IT-Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*.

Carlson, J., & Lisper, B. (2010). A resource-efficient event algebra. *Science of Computer Programming*.

Chen, J., DeWitt, D. J., Tian, F., & Wang, Y. (2000). Niagaracq: A scalable continuous query system for internet databases. In *Sigmod.*

Cox, R. (2007). Regular expression matching can be simple and fast. *URL: https://swtch.com/rsc/regexp/regexp1.html*.

Cugola, G., & Margara, A. (2009). Raced: an adaptive middleware for complex event detection. In *Middleware.*

Cugola, G., & Margara, A. (2010). Tesla: a formally defined event specification language. In *Debs.*

Cugola, G., & Margara, A. (2012a). Complex event processing with t-rex. *The Journal of Systems and Software*.

Cugola, G., & Margara, A. (2012b). Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR).*

Demers, A., Gehrke, J., Hong, M., Riedewald, M., & White, W. (2006). Towards expressive publish/subscribe systems. In *Edbt.*

Frougny, C., & Sakarovitch, J. (1993). Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*.

Galton, A., & Augusto, J. C. (2002). Two approaches to event definition. In *Dexa.*

Gatziu, S., Fritschi, H., & Vaduva, A. (1996). *Samos: an active object-oriented database system* (Tech. Rep.).

Golab, L., & Özsu, M. T. (2003). Issues in data stream management. *Sigmod Record*.

Groover, M. P. (2007). *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to automata theory, languages and computation*.

Ikonomovska, E., & Zelke, M. (2013). Algorithmic techniques for processing data streams. *Dagstuhl Follow-Ups*.

Luckham, D. (1996). *Rapide: A language and toolset for simulation of distributed systems by partial orderings of events.*

Mansouri-Samani, M., & Sloman, M. (1997). Gem: A generalized event monitoring language for distributed systems. *Distributed Systems Engineering*.

McCarthy, D., & Dayal, U. (1989). The architecture of an active database management system. *SIGMOD Record*.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*.

Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. *IEEE network*.

Paton, N. W., & Díaz, O. (1999). Active database systems. *ACM Computing Surveys (CSUR)*.

Pietzuch, P., Shand, B., & Bacon, J. (2003). A framework for event composition in distributed systems. In *Middleware.*

Ramakrishnan, R., & Gehrke, J. (2003). *Database management systems (3. ed.)*. McGraw-Hill.

Sahay, B., & Ranjan, J. (2008). Real time business intelligence in supply chain analytics. *Information Management & Computer Security*.

Sakarovitch, J. (2009). *Elements of automata theory*. Cambridge University Press.

Schultz-Møller, N. P., Migliavacca, M., & Pietzuch, P. (2009). Distributed complex event processing with query rewriting. In *Debs.*

Segoufin, L. (2006). Automata and logics for words and trees over an infinite alphabet. In *Csl.*

Seidl, H. (1990). Deciding equivalence of finite tree automata. *SIAM Journal of Computing*.

Veanes, M. (2013). Applications of symbolic finite automata. In *Ciaa* (pp. 16–23). Springer.

Wu, E., Diao, Y., & Rizvi, S. (2006). High-performance complex event processing over streams. In *Sigmod.*

Zhang, H., Diao, Y., & Immerman, N. (2014). On complexity and optimization of expensive queries in complex event processing. In *Sigmod.*

Zimmer, D., & Unland, R. (1999). On the semantics of complex events in active database management systems. In *Icde.*