Scalable Data Analytics,
Scalable Algorithms, Software Frameworks
and Visualization ICT-2013 4.2.a

Project **FP7-619435/SPEEDD**
Deliverable **D3.2 (revised)**
Distribution **Public**

http://speedd-project.eu

Second version of event recognition and forecasting technology – Part I

Fabiana Fournier (IBM) and Inna Skarbovsky (IBM)

Status: Revised (Version 2.0)

August 2016

Project

Project Ref. no	FP7-619435
Project acronym	SPEEDD
Project full title	Scalable ProactivE Event-Driven Decision Making
Project site	http://speedd-project.eu/
Project start	February 2014
Project duration	3 years
EC Project Officer	Stefano Bertolo

Deliverable

Deliverable type	Prototype
Distribution level	Public
Deliverable Number	D3.2
Deliverable Title	Second version of event recognition and forecasting technology
Contractual date of delivery	M22 (November 2015)
Actual date of delivery	August 2016
Relevant Task(s)	WP3/Tasks 3.2-3.3
Partner Responsible	IBM
Other contributors	NCSR
Number of pages	79
Author(s)	Fabiana Fournier (IBM) and Inna Skarbovsky (IBM)
Internal Reviewers	Ivo Correia (Feedzai) and Elias Alevizos (NCSR)
Status & version	Revised
Keywords	Complex event processing, forecasted/predicted event, event recognition, uncertain event

Executive summary

This document is the first part of Deliverable 3.2 “Second version of event recognition and forecasting technology” and its purpose is to present the intermediate results of T3.2 (event recognition under uncertainty) and T3.3 (event forecasting under uncertainty), and the second version of the event recognition and forecasting component (software).

A new version of the event-driven application for the two use cases of the project, the credit card fraud detection and the traffic management, were developed, implemented, and tested during the second year of the project. The full description of the applications is detailed along with evaluation tests performed. As a result of the performance evaluation, a root cause analysis has been performed and improvements in the run-time engine have been incorporated.

Our main findings from both applications show the latent benefits from the inclusion of uncertainty aspects in both applications. In the fraud detection use case – we are able to derive a potential fraud before reaching the threshold determined and therefore, have the chance to block the credit card (alternatively, deny the next transaction) before further transactions take place. In the traffic use case - we are able to forecast a congestion before it actually happens and therefore proactive actions can be taken to alleviate the congestion.

Document history

Version	Date	Author	Change Description
0.1	31/10/2015	Fabiana Fournier (IBM)	First draft
0.2	15/11/2015	Fabiana Fournier (IBM)	Second draft for internal review
0.3	25/11/2015	Fabiana Fournier (IBM)	Updates per internal review
1.0	30/11/2015	Fabiana Fournier (IBM)	Final fixes and cleanup
2.0	17/07/2016	Fabiana Fournier (IBM)	Revised version upon second annual review comments
3.0	25/07/2016	Fabiana Fournier (IBM)	Final version after internal review

Table of contents

List of tables

List of figures

Acronyms

API	Application Programming Interface
CEP	Complex Event Processing
CNP	Card Not Present
CP	Card Present
CVV	Card Verification Value
EPA	Event Processing Agent
EPN	Event Processing Network
JSON	JavaScript Object Notation
PETITE	Proton EvenT Injection & Time comprEssion
PROTON	PROactive Technology ONline
SPEEDD	Scalable ProactivE Event-Driven Decision making
WP	Work Package

1 Introduction

1.1 Purpose and scope of the document

Work Package 3 (WP3) "Real-Time Event Recognition and Forecasting under Uncertainty" deals with all the developments around event processing technologies under uncertainty. This report is the second version of SPEEDD (Scalable ProactivE Event-Driven Decision) event recognition and forecasting technology and it includes intermediate results for T3.2 (event recognition under uncertainty) and T3.3 (event forecasting under uncertainty), and the second version of the event recognition and forecasting component (software). This report presents new advanced implementations to the project two use cases: the traffic management use case and the credit card fraud use case. Basically, this report covers the extensions made to the PROactive Technology ONline (PROTON) CEP tool and the new implementations for the two use cases.

As the main goal of the CEP component is to demonstrate the usefulness of taking into account uncertainty aspects in event-driven applications, we demonstrate the applicability and benefits of the inclusion of uncertainty aspects by comparing two identical applications for each of the use cases, one applying uncertainty and the other without taking uncertainty aspects into account.

This report also describes our preliminary results for integration of the rules learnt by the machine learning component in WP3 by a semantic translation of a sample rule set from event calculus to PROTON's semantic language.

This report is structured as follows: Section 2 provides some essential concepts and terms used throughout this report. Section 3 describes extensions made to the PROTON CEP tool during year 2 of the project. Sections 4 and 5 constitute the core part of this report and describe the developments concerning the credit card fraud and traffic management use cases respectively. We present new patterns and evaluation tests for each of the use cases. In Section 7 we present our initial findings in translating event calculus semantics (event rules learnt by the machine module in SPEEDD) to PROTON's programming model. We conclude the report with summary and future steps.

1.2 Relationship with other documents

At the heart of the SPEEDD prototype resides the complex event processing component, therefore, this report is strongly related to D6.3 "First Integrated prototype" and D6.4 "Second Integrated Prototype" (under preparation to be submitted by the end of the second year of the project). The requirements for the CEP engine are dictated from the use cases in the project, thus, this report is also strongly related to system requirements for the Proactive Traffic Management use case described in D8.1 and for the Proactive Credit Card Fraud Management described in D7.1. With relation to the traffic management use case, this report is also related to the developments in the micro simulator, and therefore related to

D8.2 and D8.4, First (submitted at month 12) and Final (to be submitted at month 24) Version of Micro-Simulator respectively.

The main goal of the CEP component is to derive forecasted events that feed the decision making component so actions can be taken before an undesired event (such as a congestion situation in the high way) takes place. Therefore our work is also related to D4.2 "Second version of real-time decision-making technology".

1.3 Updates since the first version

This second revision of the document contains two main additions with regards to its first version:

First, a performance analysis of throughput and latency that drove a further investigation on possible bottlenecks has been performed and described in Section 6. Furthermore, improvements to the run-time engine have been introduced as a result of this analysis.

Second, to better assess the quality of our patterns, especially, the inclusion of uncertainty aspects in the application; we performed a recall and precision analysis on the traffic use case (Section5.2.2.3).

2 Preliminaries

Each complex event processing engine uses its own terminology and semantics. We follow the semantics presented in Etzion's and Niblet's book [1]. In our previous deliverable (D6.1 – First version of event recognition and forecasting technology[1]) we covered the main constructs of the CEP component in both the design and run-time and described the main extensions incorporated to PROTON to cope with uncertainty. For the sake of clarity of this report, we only briefly mention below again two main concepts: Event processing Network (EPN) and pattern matching process. For further concepts and details on PROTON refer to[2]

2.1 Event Processing Network (EPN)

An **Event Processing Network (EPN)** is a conceptual model, describing the event processing flow execution. An EPN comprises a collection of Event processing Agents (EPAs), event producers, events and consumers (Figure 1). The network describes the flow of events originating at event producers and flowing through various event processing agents to eventually reach event consumers. For example, in Figure 1, events from Producer 1 are processed by EPA 1. Events derived by EPA 1 are of interest to Consumer 1 but are also processed by EPA 3 together with events derived from EPA 2.

[1] Available at http://speedd-project.eu/press_room
[2] https://github.com/ishkin/Proton/tree/master/documentation

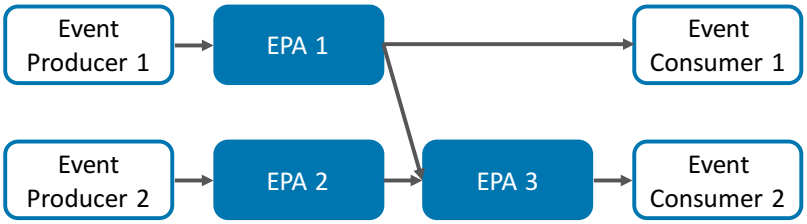

Figure 1: Illustration of an event processing network

2.2 Pattern matching process

An EPA performs three logical steps, a.k.a **pattern matching process** or **event recognition** (see Figure 2). Please note that all three steps are optional but at least one must be done inside an EPA.

- The **filtering step**, in which relevant events from the input events are selected for processing according to the filter conditions. The output of this step is a set of **participant events**.
- The **matching step** that takes all events that passed the filtering and looks for matches between these events, using an event processing pattern or some other kind of matching criterion. The output of this step is the **matching set**.
- The **derivation step** that takes the output from the matching step and uses it to derive the output events by applying derivation formulae.

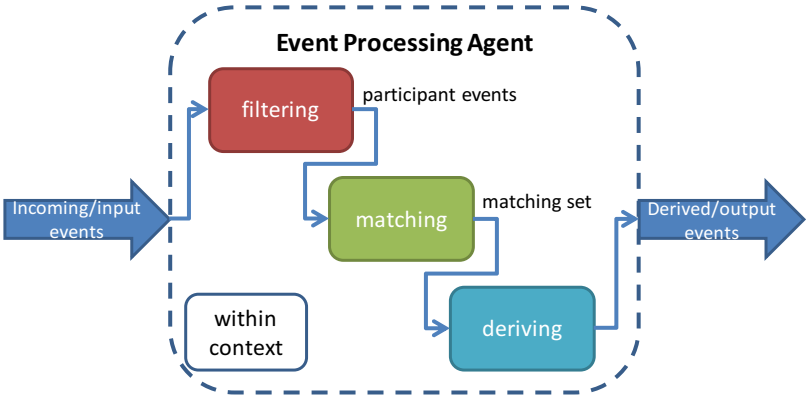

Figure 2: Event recognition process in an EPA

An **event pattern** is a template specifying one or more combinations of events. Given any collection of events, if it's possible to find one or more subsets of those events that match a particular pattern, it can be said that such a subset satisfies the pattern. Some common examples of patterns applied in our scenarios:

- **Sequence**, means that at least one instance of all participating event types must arrive in a specified order for the pattern to be matched.
- **Count**, means that the number of instances in the participant event set satisfies the pattern's number assertion.

- **All**, means that at least one instance of all participating event types must arrive for the pattern to be matched; the arrival order in this case is immaterial.
- **Trend**, events need to satisfy a specific change (increasing or decreasing) over time of some observed value; this refers to the value of a specific attribute or attributes.
- **Sum**, means that the value of a specific attribute, summed up over all participant events, satisfies the sum threshold assertion.
- **Average (AVG)**, means that the value of a specific attribute, averaged over all participant events, satisfies the average threshold assertion.

In addition, PROTON also supports "extended EPAs". The idea is to embed a certain functionality (e.g., formula or model) to be then used in the EPN. An example of such extended EPA is EPA0 in the traffic management use case (see Section 5.1).

3 Extensions made to PROTON

During the second year of the project, three extensions main were made related to the CEP component:

1. Possibility of deriving attributes of the matching set in the derived event. In previous versions of PROTON the derived event didn't include attributes from the matching set events in its payload. We needed this capability in order to:
    a. Access information of the transactions that caused a credit card fraud derived event (e.g., transaction number, ATM, and location) in the credit card fraud use case. This information is used by fraud operators in their analysis of the results.
    b. Access information of the sensor readings (e.g., density and speed) that caused the predicted congestion derived event in the traffic management use case. This information is used by the traffic operator while making an operational decision in the control room at Grenoble.
2. Adding "gradient" to the TREND operator. In certain situations, such as when predicted a congestion, it is not enough to look for a decreasing or increasing pattern, but also for the gradient formed by consecutive events in the TREND pattern. In our example, a higher gradient can indicate a faster build-up of congestion. i.e., a higher probability of congestion.
3. PETITE (Proton EvenT Injection & Time compression) utility: One of the challenges in SPEEDD is to demonstrate the capability of processing and running vast amounts of input events. Therefore, we needed a mechanism that can take the input events and inject them into the engine at very high rates (much higher than in reality imposed by their timestamps) and process them very fast without altering the logic of the application. The PETITE (Proton EvenT Injection & Time compression) script, is an external utility to PROTON that given an ordered input file and a total elapsed time for processing all the events in the input file (a compression "ratio") it:
    a. Checks for feasibility of the requested ratio. This means that events are injected in a way that can be processed by the engine without being out of order, contexts are not too small, and there is a minimal interval between input events. For example, having an

input file that starts with the first input event occurrence time at Dec.1 2014 and ends with the last input event occurrence time at Dec. 7 2014, we target to run all these events in only one hour then our ratio is 168 (we need to "compress" or reduce all times by 168, as we are moving from one week or 168 hours to one hour).

b. Alters the temporal contexts in the JSON definition file for PROTON in such a way that the ratio is met and the application logic is maintained.
c. Alters the time intervals between input events so that they are compressed to the required injection rate. The original date and time remains intact for the purposes of date comparisons and manipulation, however, an additional column of timestamps are added to the input file, so that they can be used as "OccurenceTime" attribute in the Timed File Input adapter of PROTON for injection of events based on those intervals.

The first two extensions are related to the CEP run-time engine and UI and are reflected in the updated EPNs of the use cases, whilst the third one is related to a utility that can be applied once the JSON (JavaScript Object Notation) of the application is created for testing purposes before the application is deployed. The details of the extensions to the engine can be found under the documentation folder in PROTON's open source repository[3] . The PETITE utility along with a README file are also stored in the open source repository of PROTON at[3].

4 Event processing application for the credit card fraud detection use case

In this section we describe the work carried out in the fraud detection use case including the design of the event processing network, the implementation of the application, and its evaluation using real-data.

4.1 Design of second EPN for the credit card fraud detection use case

As in the previous deliverable in the fraud use case, we distinguish between two types of transactions: CP (Card Present) and CNP (Card Not Present). Most of the event patterns are common to both cases (see below). What differs is the temporal time windows length (CNP is much faster so the temporal windows are shorter).

A second version of the EPN for the fraud detection use case that includes improvements and insights gained during year one was devised during the second year of the project. The resulting EPN consists of 18 EPAs. For the sake of clarity, we treat the CP and CNP case as two separate EPNs (CP-EPN and CNP-EPN) as shown in Figure 3 and Figure 4 respectively and detailed in the following sections. For the sake of simplicity we only show the EPAs and the events flow in the network in the Figures. Yellow dotted lines represent events, other than input events, that are initiators of a context. The PROTON JSON definitions file that comprises the application is provided as part of the software deliverable that accompanies this report.

[3] https://github.com/ishkin/Proton/

Note EPAs numbered 8, 10-12. These are combination of patterns, that is, a derived event of one pattern is the temporal context initiator of another EPA. In other words, two patterns occur one after the other, and derive a new event with higher probability of fraud than the original derived event (of the single EPA). The assumption is that when two patterns occur one after the other, the probability of a fraud is higher than in the separate EPAs. The sequencing between patterns is done by causing one derived event to open the temporal context of another EPA. The new derived event (of the latter new EPA), has the same probability of the original EPA + 0.1 or 1 depending on the pattern. In other words, the new EPA looks the same as the original EPA except for two differences: the probability of the derived event is higher, and the context initiator is the derived event of the first EPA. Other policies remain the same. The addition of 0.1 was selected to reflect a larger certainty value of a fraud as two patterns need to occur in a row compared to each pattern separately. However, other "small values" can be selected and tested as well.

In the CP-EPN, situations marked as potential (uncertain) frauds are fired in the following cases Figure 3:

- Consecutive withdrawals of increasing or decreasing amounts for a single card (EPA1, *IncreasingAmounts* and EPA2, *DecreasingAmounts*).
- A high number of transactions in a short time window for a single card (EPA3, *FlashAttack*).
- Many large withdrawals from a single ATM (EPA4, *MultipleMaxATMWithdrawals*).
- Sudden card use near the expiration date (EPA5, *SuddenCardUseNearTheExpirationDate*).
- Consecutive attempts to use the same card in different physical locations (EPA6, *TransactionsInFarAwayPlaces*).
- Small amount followed by big purchase for a single card (EPA7, *SmallAmountFollowedByBigPurchase*).
- A *FlashAttack* occurs after having the sequence of a big purchase after a small one. (EPA8, *FlashAttackAfterSmallFollowedByBigAmounts*)

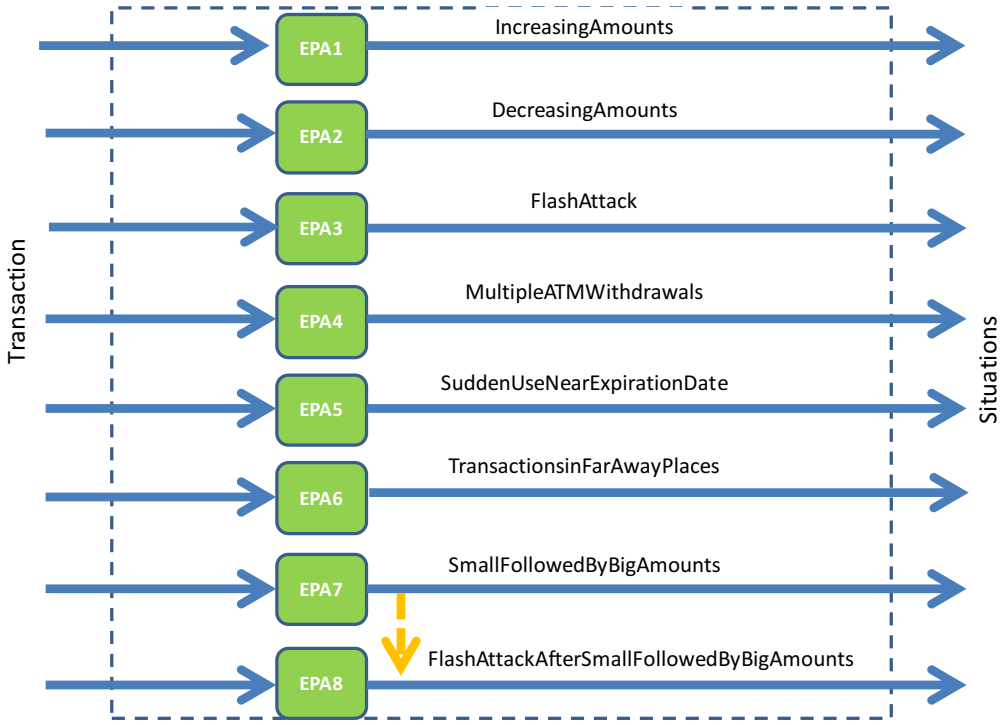

Figure 3: Credit card fraud detection use case CP-EPN

In the CNP-EPN, in addition to the situations marked as potential (uncertain) frauds in the CP case, the following situations are fired (EPA9-EPA12 in Figure 4):

- Several attempts of using a wrong CVV (Card Verification Value) for the same card are made (EPA9, *CVVatack*).
- Increasing amounts of withdrawals/payments are carried out after several attempts of using a wrong CVV (EPA10, *IncreasingAmountsAfterCVVAttack*).
- A high number of transactions for a single card after several attempts of using a wrong CVV (EPA11, *FlashAttackAfterCVVAttack*)
- Sudden card use near the expiration date after several attempts of using a wrong CVV (EPA12, *SuddenCardUseNearExpirationDateAfterCVVAttack*)
- Multiple occurrences of a suspicious fraudulent card happen at the same ATM (EPA3 and EPA4)
- There are two consecutive attempts of using the same card in different locations (EPA7 or *ClonedCard*).

Note that EPA4 (*MultipleMaxATMWithdrawals*) and EPA6 (*TransactionsInFarAwayPlaces*) are only applicable to the CP case and therefore greyed out in Figure 4.

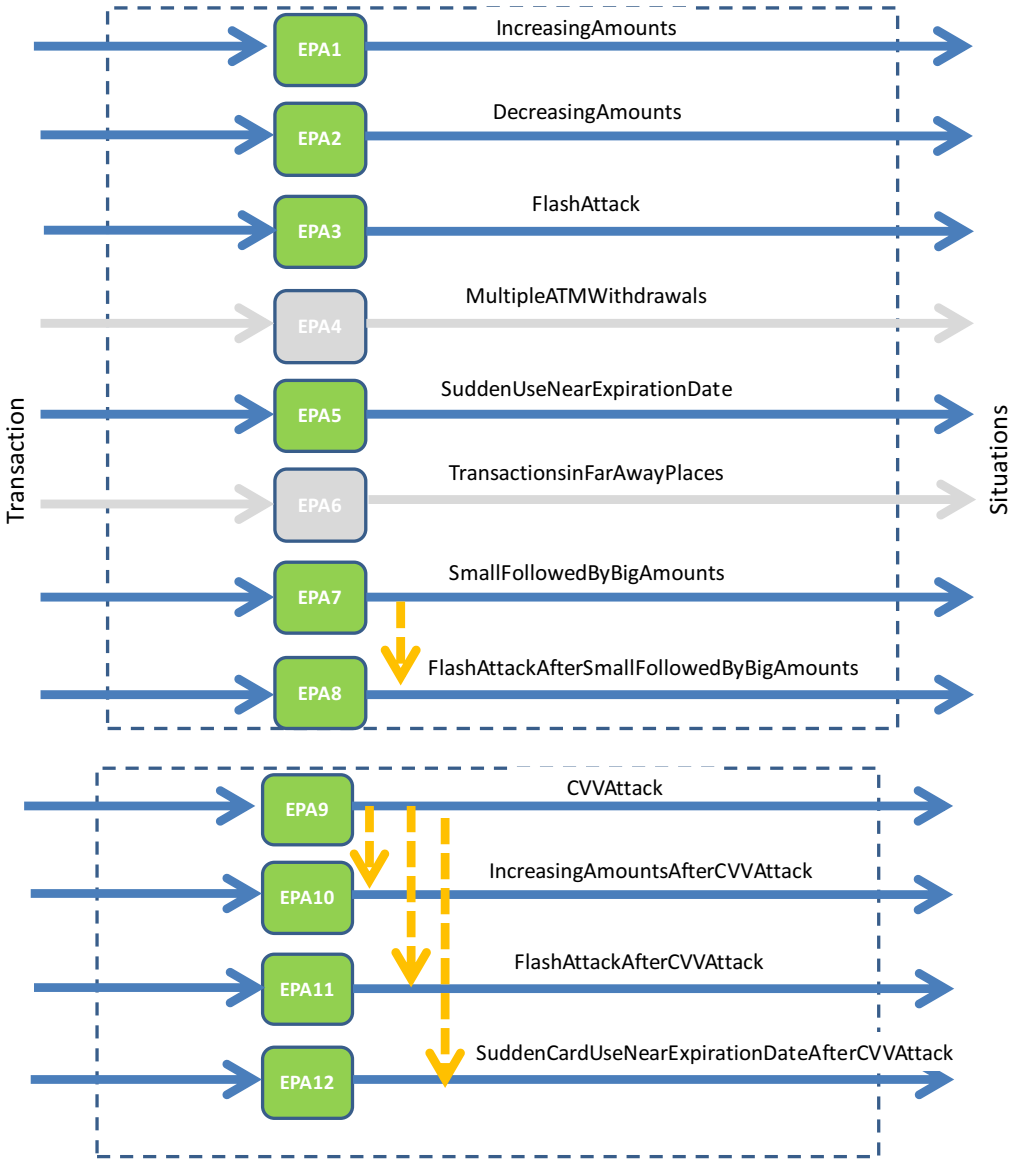

Figure 4: Credit card fraud detection use case CNP-EPN

4.1.1 Event types

Thirteen event types have been defined that comprise the event inputs, outputs/derived, and situations as shown in Table 1. For the sake of simplicity we only show the user-defined attributes or the event payload and not the metadata.

Although the names of concepts can be determined freely by the application designer in PROTON, we use some naming conventions for the sake of clarity. We denote event types with capital letters. Built-in/metadata attributes start with a capital letter, as well as payload attributes that hold operators values, while payload attributes start with a lower letter. Table 1 shows the event definitions for the fraud EPN, where:

card_pan (type: String) - The number that identifies the card. The card BIN, which corresponds to the first six digits of the PAN, can give information such as the issuer of the card.

terminal_id (Type: Long) - The internal identification of the terminal.

cvv_validation (Type: Int) - Variable indicating whether the CVV (Card Verification Value) was used or not. In the positive case, it indicates whether it was valid or not. This three digit number printed on the signature panel on the back of the card helps to verify authorized possession of a credit card. For testing purposes we use *validation_code* == 16 as we don't have the real code number from the real data.

amount_eur (Type: Double) - The amount in euros of the transaction.

acquirer_country (type: Int) - The country of the acquiring bank.

is_cnp (Type: Bit) - Flag that states whether the transaction happened in the CP or CNP context.

card_exp_date (type: Date (YYYYMM)) - The expiration date of the card.

Note that the *Transaction* raw event includes more fields or attributes. We defined only the ones required for pattern detection in the current EPN implementation. When running in SPEEDD architecture, PROTON ignores event attributes not specified in its JSON. Also note that [matching_set] indicates the set (array) of events that constitute the matching set of the specific pattern. For example, [matching_set] in *IncreasingAmounts* indicates the events that satisfy the TREN pattern.

Table 1: Event types for the credit card fraud detection use case

Event name	Transaction
Payload	card_pan; terminal_id; cvv_validation; amount_eur; acquirer_country; is cnp; card_exp_date
Event name	IncreasingAmounts
Payload	card_pan; TrendCount; is_cnp; [matching_set]
Event name	DecreasingAmounts
Payload	card_pan; TrendCount; is_cnp; [matching_set]
Event name	CVVAttack
Payload	card_pan; TransactionsCount; is_cnp; [matching_set]
Event name	FlashAttack
Payload	card_pan; TransactionsCount; is_cnp; [matching_set]
Event name	SmallFollowedByBigAmounts
Payload	card_pan; is_cnp; [matching_set]
Event name	MultipleATMWithdrawals
Payload	terminal_id; TransactionsCount; [matching_set]
Event name	SuddenUseNearExpirationDate
Payload	card_pan; TransactionsCount; is_cnp; card_exp_date; [matching_set]
Event name	TransactionsinFarAwayPlaces
Payload	card_pan; [matching_set]
Event name	FlashAttackAfterSmallFollowedByBigAmounts

Payload	card_pan; TransactionsCount; is_cnp; [matching_set]
Event name	IncreasingAmountsAfterCVVAttack
Payload	card_pan; TrendCount; is_cnp; [matching_set]
Event name	FlashAttackAfterCVVAttack
Payload	card_pan; TransactionsCount; is_cnp; [matching_set]
Event name	SuddenCardUseNearExpirationDateAfterCVVAttack
Payload	card_pan; TransactionsCount; is_cnp; card_exp_date; [matching_set]

4.1.2 Event processing agents

Henceforth, we describe the EPAs in the following order: Event name; motivation; event recognition process (following Figure 2); contexts along with temporal context policy; and pattern policies.

In the event recognition process we only show the steps that take place in the specific EPA, while the others are greyed. For the *filtering step* we show the filtering expression; for the *matching step* we denote the pattern variables; and for the *derivation step* we denote the values assignment and calculations. Please note that for the sake of simplicity we only show the assignments that are not copy of values (all other derived event attributes values are copied from the input events). For attributes, we just denote their names without the prefix of 'attribute_name.'

As during the first year, we apply the *Sigmoid* or *Logistic* probabilistic function in our pattern derivations. . A Sigmoid function[4] receives three inputs (*a,b,x*) and returns

$$\frac{1}{1+e^{-(a+bx)}}$$

The *x* is the pattern parameter value such as *count* for the COUNT pattern. The values for *a* and *b* in each pattern were calculated based on the targeted probability. For example, assuming that we would like to derive a probability of 0.6 for x=4 and 0.7 for x=5, what are then the values for *a* and *b*? As in the patterns, the targeted probabilities were given to us by the domain expert from Feedzai. The selected values shown in the EPAs for the pattern aggregators and the coefficients in the Sigmoid function are given in Table 2.

4.1.3 CP-EPAs

4.1.3.1 EPA1: IncreasingAmounts

Motivation: EPA1 is a pattern that can be used to test the system detection limits. It checks for at least four consecutive valid transactions (i.e., trendN > 3) with increasing amounts.

[4] See for example: http://www.stat.ubc.ca/~rollin/teach/643w04/lec/node46.html

Event recognition process:

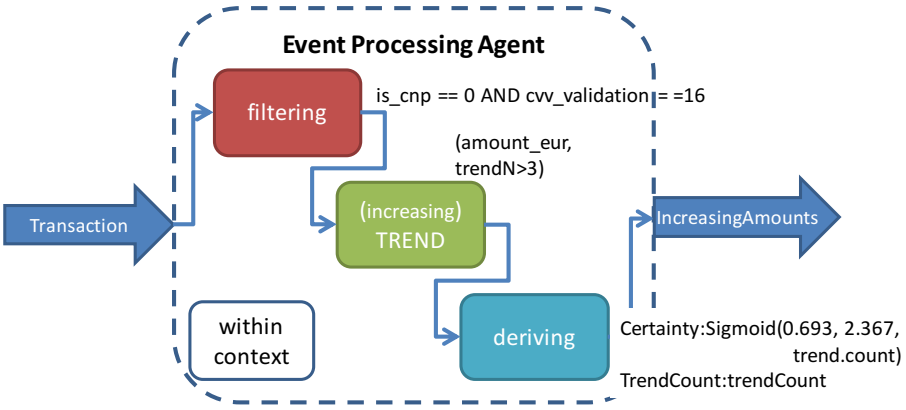

Figure 5: Event recognition process for IncreasingAmounts EPA

trendNumber and *trend.count* are built-in TREND variables that denote the minimal number of input events required in order to satisfy the pattern and the actual number correspondingly.

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A temporal window of 15 minutes opens with the arrival of a first *Transaction* event. In this elapsed time we check for a Trend pattern over the amounts in the transactions per card. According to the policies selected, we will derive a new event every time the (increasing) Trend pattern is satisfied (having higher probabilities values in the *Certainty* attribute). In the derived event we also report the actual number of transactions that satisfy the Trend pattern (by the built-in *trendCount* attribute).

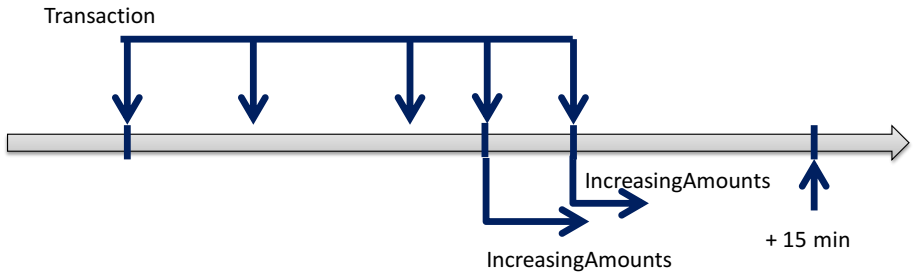

Figure 6: Context for DecreasingAmounts EPA

4.1.3.2 EPA2: DecreasingAmounts

Motivation: This pattern is similar to EPA1, as it aims to test system detection limits. EPA2 checks for at least four consecutive valid transactions; that is, correct CVV, with decreasing amounts that passed the card limit. The idea is that by passing the card limit, the fraudster is obliged to decrease the amount of money in the following transaction. The derived event is *DecreasingAmounts*. The contexts and policies are the same as in EPA1. For the current implementation we apply a standard average card limit in Europe (400 euros).

Event recognition process:

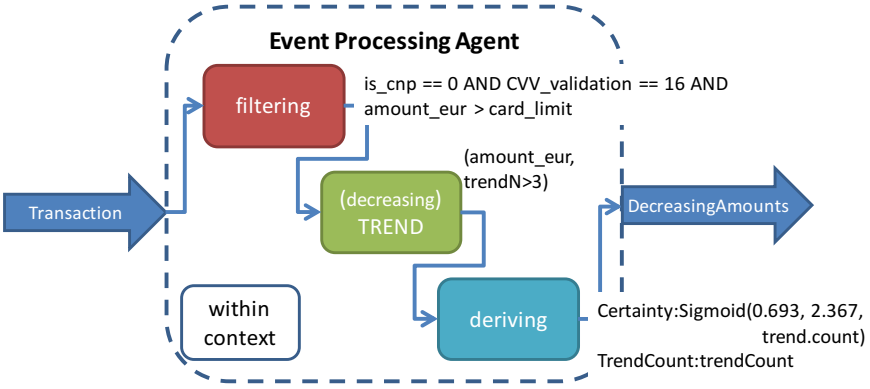

Figure 7: Event recognition process for DecreasingAmounts EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A temporal window of 15 minutes opens with the arrival of a first *Transaction* event. In this elapsed time we check for a Trend pattern over the amounts in the transactions per card. According to the policies selected, we will derive a new event every time the (decraesing) Trend pattern is satisfied (having higher probabilities values in the *Certainty* attribute). In the derived event we also report the actual number of transactions that satisfy the Trend pattern (by the built-in *trendCount* attribute).

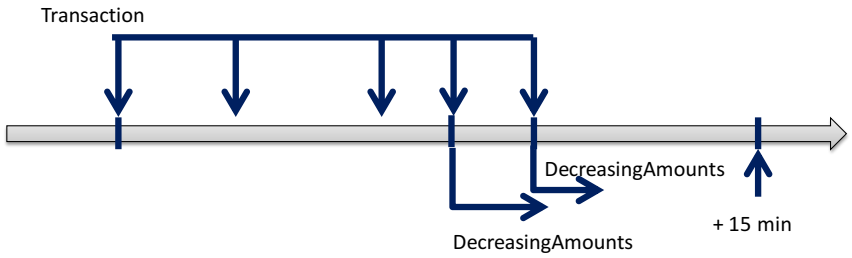

Figure 8: Context for DecreasingAmounts EPA

4.1.3.3 EPA3: FlashAttack

Motivation: In EPA3 we look for a high number of transactions in a short period of time.

Event recognition process:

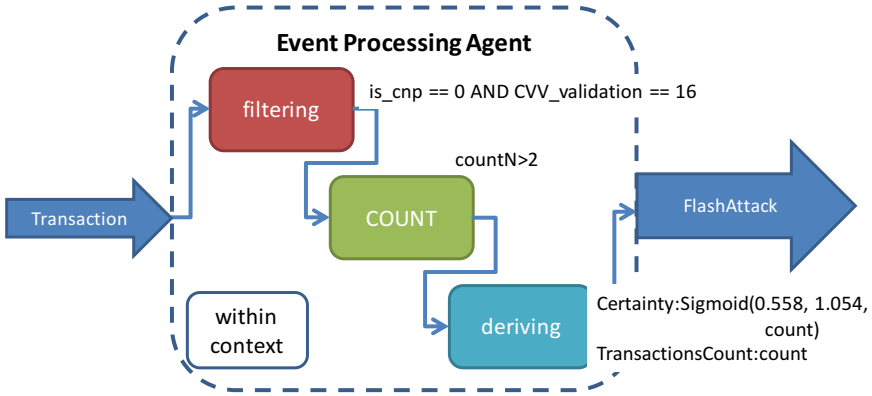

Figure 9: Event recognition process for FlashAttack EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A temporal window of 7 min is opened with the arrival of a first *Transaction* event. In this elapsed time we COUNT the transactions per a single card. If the number of events is higher than 3, then we derive a single event at the end of the window with the number of events (*count*) at the moment of derivation

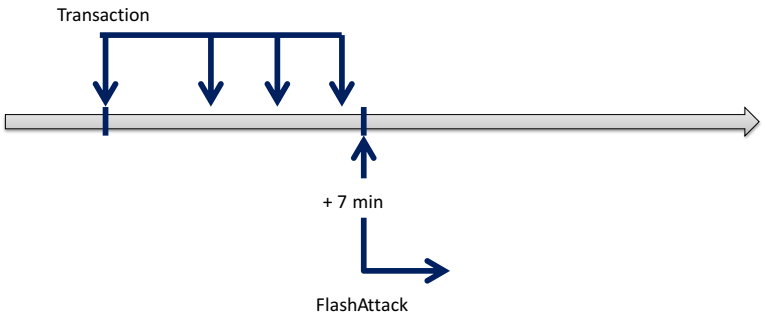

Figure 10: Context for FlashAttack EPA

4.1.3.4 EPA4: MultipleMaxATMWithdrawals

Motivation: Given that ATMs have an upper limit for withdrawals, in this kind of attack, fraudsters are simply trying to take as much money as they can in the fewest possible transactions. A typical large value in an ATM in Europe is 200 euros. We look for at least three large withdrawals in a single ATM. This EPA holds only for the CP case.

Event recognition process:

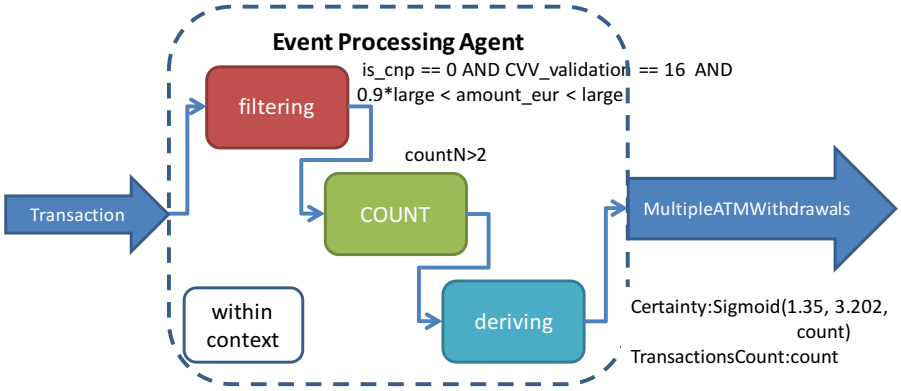

Figure 11: Event recognition process for MultipleMaxATMWithdrawals EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by terminal_id

Initiator policy: IGNORE

Meaning: We derive a new event whenever the count pattern is satisfied within a ten minute window.

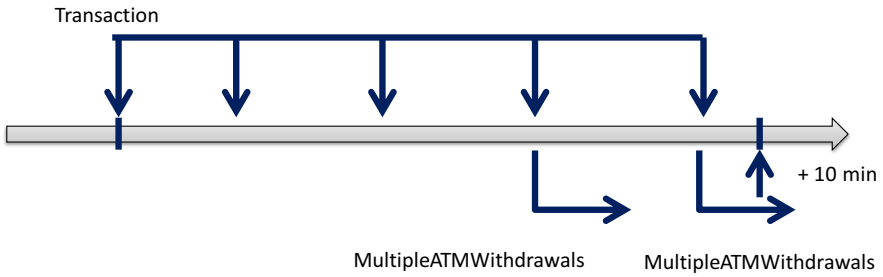

Figure 12: Context for MultipleATMWithdrawals EPA

4.1.3.5 EPA5: SuddenCardUseNearEexpirationDate

Motivation: Fraudsters may obtain credit card credentials, and then sell them to other people. When the expiration date approaches, and they cannot sell those cards, they will try to make as much profit as they can from those cards before they lose control over them. We look for frequent transactions within a relatively short period of time (20 min) for a single card on the day of, or the day before, the card expiration date.

Event recognition process:

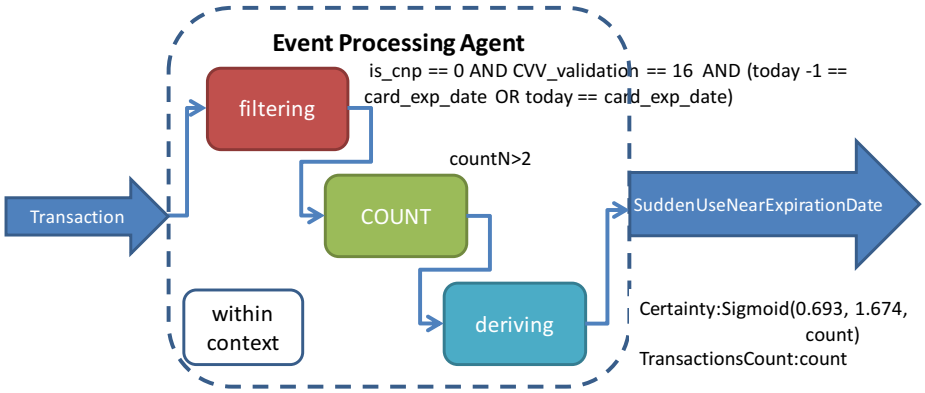

Figure 13: Event recognition process for SuddenCardUseNearEexpirationDate EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A temporal window of 20 min is opened with the arrival of a first *Transaction* event per card. During the time window, we look for transactions at the day or one day earlier to the card expiration date.

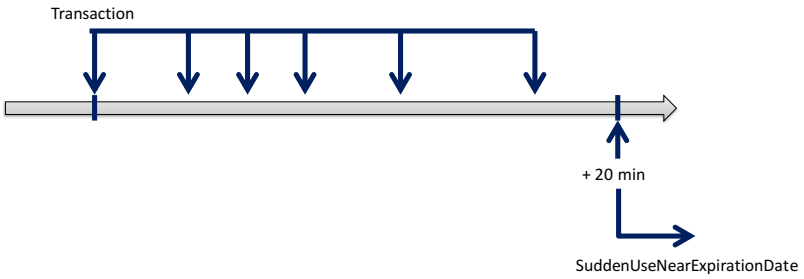

Figure 14: Context for SuddenCardUseNearEexpirationDate EPA

4.1.3.6 EPA6: TransactionsInFarAwayPlaces

Motivation: Due to speed limitations in traveling, it is impossible for the same card to be used in faraway places. This means that the card has been cloned. In this pattern, we check that two consecutive transactions for a specific card cannot be at different physical places within a short period of time. Note that we do not check for CVV validity, as we are only interested in checking for use of the same card in different places. In addition, the derived event has a certainty of 1 (in this case the fraud indication is 100%) and therefore the Certainty attribute is not shown in the deriving step (the default is "1"). T1 and T2 are aliases of the event type Transaction. This EPA holds only for the CP case.

Event recognition process:

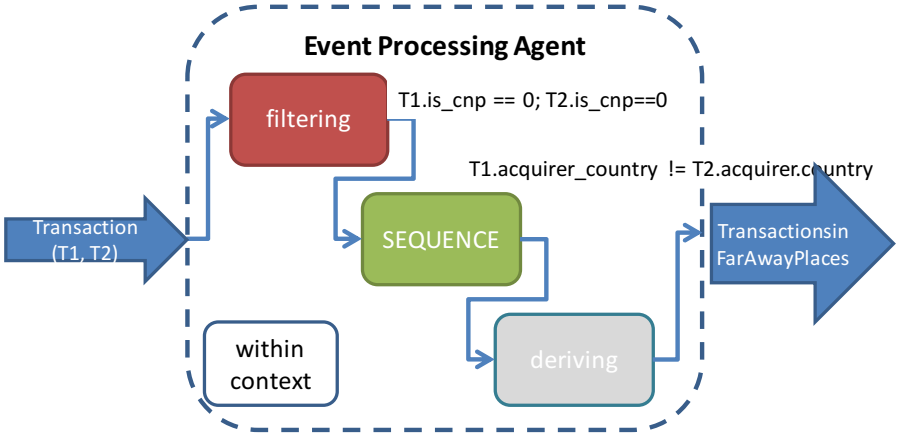

Figure 15: Event recognition process for TransactionsInFarAwayPlaces EPA

Note that the *SequenceFraud* derived event has a certainty of 1 (in this case the fraud indication is of 100%) and therefore the *Certainty* attribute is not shown in the derivation step (the default is "1").

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	T1=FIRST; T2=OVERRIDE	CONSUME

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: In a 15 minute temporal window we detect and immediately alert every attempt to use the same card for any two consecutive transactions in faraway places.

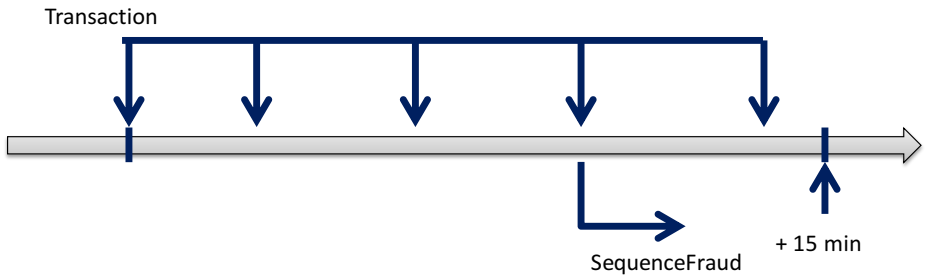

Figure 16: TransactionsInFarAwayPlaces EPA

4.1.3.7 EPA7: SmallAmountFollowedByBigAmount

Motivation: This pattern tests for system thresholds. Sometimes the fraudsters test a small amount, and once the transaction succeeds, they attempt a big purchase. We look for a sequence pattern where the first transaction is related to a very small amount (cents) and the second one in the sequence is related to a big amount (>200). Note that T1 and T2 are aliases of the event type Transaction. The *SmallAmountFollowedByBigAmount* derived event has a 0.8 probability of being a fraudulent transaction to reflect the fact that once it is detected it is almost certain that the transaction is fraudulent.

Event recognition process:

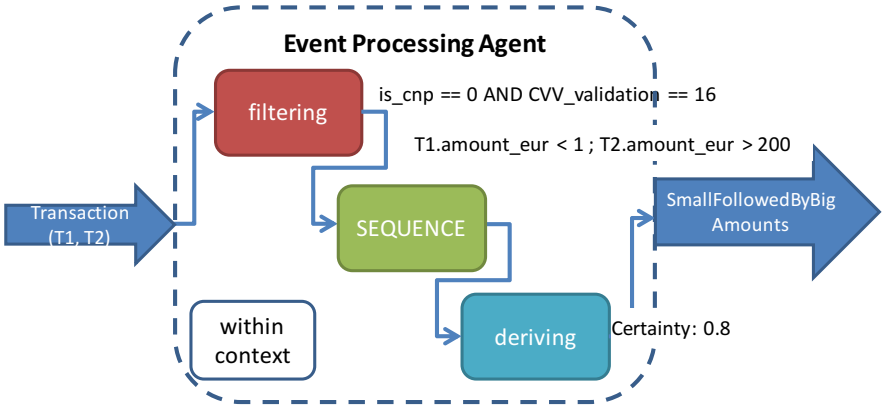

Figure 17: Event recognition process for SmallAmountFollowedByBigAmount EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	SINGLE	T1=FIRST; T2=OVERRIDE	CONSUME

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A short temporal window of 7 min opens with the arrival of a first *Transaction* event with a small amount per card. The pattern detects a small purchase, and if the consecutive transaction is a big purchase it immediately emits the situation.

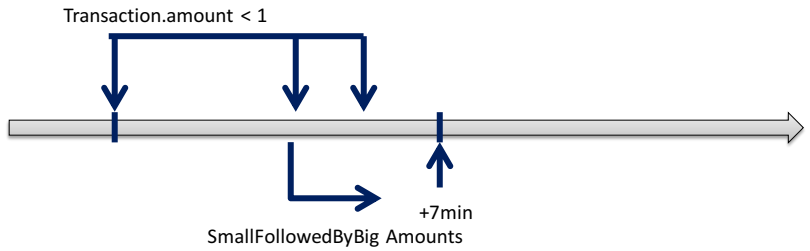

Figure 18: SmallAmountFollowedByBigAmount EPA

4.1.3.8 EPA8: FlashAttackAfterSmallFollowedByBigAmounts

Motivation: The *FlashAttackAfterSmallFollowedByBigAmounts* situation with uncertainty of 0.9 (0.8+0.1) is derived when there is a Flash attack after having the sequence of a big purchase after a small one. The context is opened with the *SmallFollowedByBigAmounts* derived event.

Event recognition process:

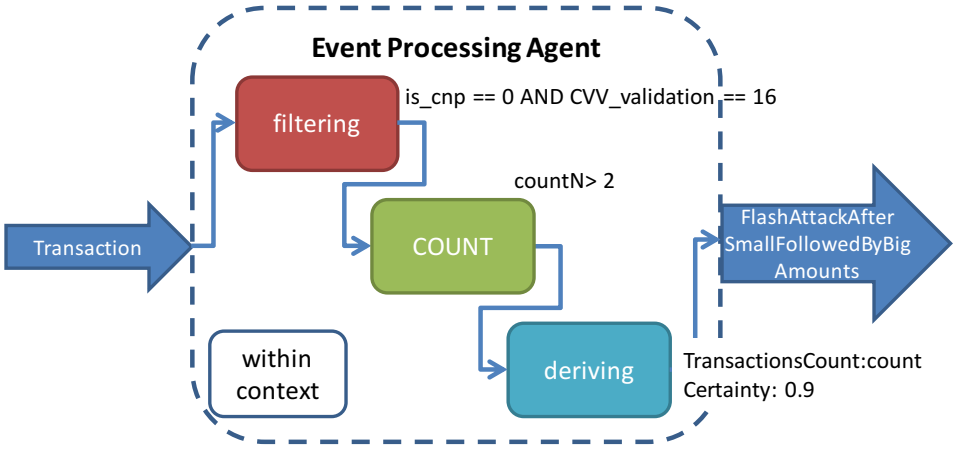

Figure 19: Event recognition process for FlashAttackAfterSmallFollowedByBigAmounts EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A temporal window of 7 min is opened with the arrival of the *SmallFollowedByBigAmounts* event event. In this elapsed time we COUNT the transactions per a single card. If the number of events is higher than 2, then we derive a single event at the end of the window with the number of events (*count*) at the moment of derivation.

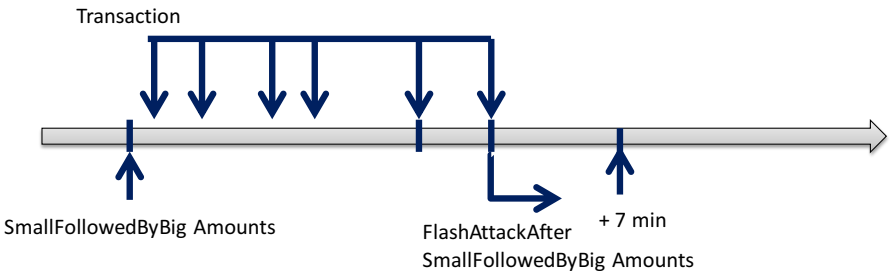

Figure 20: Context for FlashAttackAfterSmallFollowedByBigAmounts EPA

4.1.4 CNP-EPAs

The CNP-EPN includes the counterpart EPAs for 1-3, 5, and 7-8 where:

- The temporal windows are shorter (only a couple of minutes), as on-line transactions are much faster
- Filter condition *is_cnp ==1* (instead of *is_cnp ==0*).

In addition, it includes EPA9 and combined EPAs 10-12 as detailed below.

4.1.4.1 EPA9: CVVAttack

Motivation: The fraudsters may only have access to partial information about the card. Therefore, to obtain the rest of the information, such as CVV, they can do a scan over possible CVV values. In this EPA, we look for cases in which at least four attempts with incorrect CVV are made in a very short period of time (two minutes).

Event recognition process:

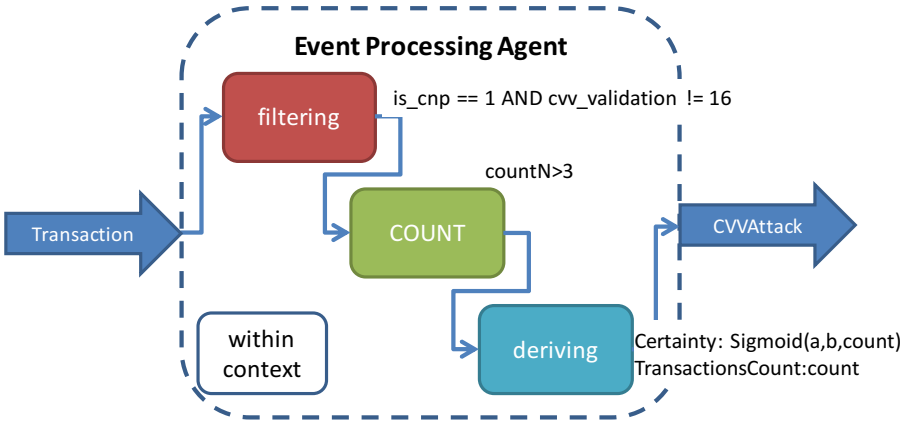

Figure 21: Event recognition process for CVVAttack EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by card_pan

Initiator policy: IGNORE

Meaning: A short temporal window of 2 min is opened with the arrival of a first *Transaction* event per card. At the end of the window the COUNT evaluation is made and a derived event is emitted if the pattern is satisfied.

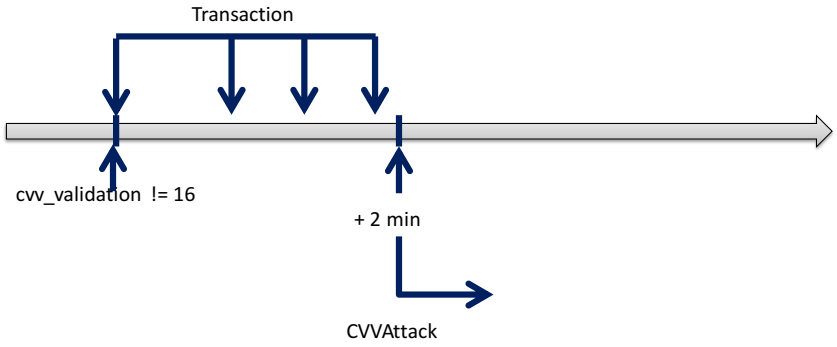

Figure 22: Context for CVVAttack EPA

4.1.4.2 EPA10-EPA12: Combined Patterns

As already mentioned, the assumption is that when two patterns occur one after the other, the probability of a fraud is higher than in the separate EPAs. The sequencing between patterns is done by causing one derived event to open the temporal context of another EPA. The new derived event (of the

latter new EPA), has the same probability of the original EPA + 0.1. In other words, the new EPA looks the same as the original EPA except for two differences: the probability of the derived event is higher by 0.1, and the context initiator is the derived event of the first EPA. Other policies remain the same. In EPAs 10-12, we assume that after several "unsuccessful" attempts with wrong CVV, a "successful" attempt is made (CVV is correct) so the second pattern can be detected.

In EPA10, the *IncreasingAmountsAfterCVVAttack* situation is derived when an Increasing Amount pattern occurs after several attempts with the wrong CVV. The context is opened with the *CVVAttack* derived event.

In EPA11, the *FlashAttackAfterCVVAttack* situation is derived when a Flash Attack occurs after several attempts with the wrong CVV. The context is opened with the *CVVAttack* derived event.

In EPA12, the *SuddenCardUseNearExpirationDateAfterCVVAttack* situation is derived when a Sudden Card Use Near Expiration Date pattern occurs after several attempts with the wrong CVV. The context is opened with the *CVVAttack* derived event.

4.2 Implementation and evaluation of second EPN for the fraud detection use case

The main goal of the CEP component with regards to the credit card fraud detection use case is to assess whether the event patterns can indeed detect fraudulent situations. Eventually, we are looking to answer two fundamental questions:

- Can we detect fraudulent situations based on the implemented event patterns?
- Moreover, can we get better results when including uncertainty aspects?

The main challenge with respect to the implementation and evaluation of the event-driven application for the fraud use case is the use of real data due to privacy issues. We first tested our EPN with anonymized data provided by Feedzai. However, we realized that due to the heavy anonymization that the data went through the data didn't preserve the business logic coherence and therefore we cannot rely on the recall and precision of our findings. The only way to overcome this limitation and address the first question was to run the application using real-data at Feedzai's premises.

The way to address the second question is to have two applications or EPNs, once including uncertainty aspects and the other one without uncertainty, i.e. deterministic, and compare between the results of these two EPNs. This is a common approach in CEP engines dealing with uncertainty (see for example in [5]).

4.2.1 Implemented EPN

As explained above we needed to test our application by Feedzai's people at their premises twice: once including uncertainty (the "uncertain case") and once without the inclusion of uncertainty aspects (the "certain case") as detailed below.

As this process included several iterations over learning how to deploy PROTON and run it differently in each case, without the loss of generality, we decided to focus first only on the CP case.

Two "installation packages" of PROTON (stand-alone version) were provided to our partner in Feedzai:

- The "uncertain case" or package - The EPN as depicted in Figure 3, that is, the JSON definition.
- The "certain EPN" or package - The same EPN but without the inclusion of uncertainty aspects, that is, all derived events have a probability of 1. The contexts remain the same but the aggregators operand values are calculated as shown in Table 3.

Along with these two packages, the following has also been provided:

- Detailed steps how to install PROTON and run it
- The PETITE script (see Section 33) along with explanations
- Sample data (for tooling testing only purposes)
- Example of an output file along with explanation on how to interpret it

4.2.1.1 Values of operator operands and coefficients of the Sigmoid function

As aforementioned, two types of tests on the CP case have been conducted: the "uncertain case" and the "certain case". For the "uncertain case", the following values for the Sigmoid variables have been chosen (Table 2). Note that we need two values for *x* (*x1* and *x2*) in order to determine the coefficients *a* and *b*, where *x* denotes the pattern operand (countN for COUNT and trendN for TREND).

Table 2: Coefficients calculation for Sigmoid function in the uncertain CP case

EPA name	x1	prob1	x2	Prob2	a	b
IncreasingAmounts	4	0.6	5	0.75	-2.367	0.693
DecreasingAmounts	4	0.6	5	0.75	-2.367	0.693
FlashAttack	3	0.65	4	0.85	-2.728	1.116
MultipleATMWithdrawals	3	0.7	4	0.9	-3.202	1.350
SuddenCardUseNearExpirationDate	3	0.6	4	0.75	-1.674	0.693

For the "certain case", the first *x* that gave a probability higher than 0.9 has been selected for the counterparts EPAs. The resulting EPAs operand values are shown in Table 3. Note that these values are required for patterns of type aggregators (*x*>= the value in the table). Therefore, for the *TransactionsInFarAwayPlaces*, *SmallAmountFollowedByBigAmount*, and *FlashAttackAfterSmallFollowedByBigAmounts* EPAs, the EPAs remain the same as in the uncertain case (SEQUENCE patterns) with probability of 1.

Table 3: Patterns operand value for the certain CP case

EPA name	x
IncreasingAmounts	7
DecreasingAmounts	7

FlashAttack	6
MultipleATMWithdrawals	5
SuddenCardUseNearExpirationDate	6
FlashAttackAfterSmallFollowedByBigAmounts	4

4.2.1.2 Input and output adapters/Producers and consumers

One of the main characteristics of CEP engines is the asynchronous way in which events are received and emitted to and out of the system. This is usually done through a publish/subscribe mechanism, with no Application Programming Interface (API) definition per-se.

PROTON's JSON file that is created at build-time contains all EPN definitions, including definitions for event types, EPAs, contexts, producers, and consumers. The physical entities representing the logical entities of producers and consumers in PROTON are adapter instances. For each producer an input adapter is defined, which defines how to pull the data from the source resource and how to format the data into PROTON's object format before delivering it to the run-time engine. The adapter is environment-agnostic, but uses the environment-specific connector object, injected into the adapter during its creation, to connect to PROTON run-time.

At run-time, the standalone CEP engine receives incoming events through the input adapters, processes these incoming events according to the definitions, and sends derived events through the output adapters. At execution, the standalone run-time engine accesses the metadata file, loads and parses all the definitions, creates a thread per each input and output adapter, and starts listening for events incoming from the input adapters (producers) and forwards events to output adapters (consumers).

Note that for the distributed implementation on top of STORM, an input Bolt serves the same function as input adapter, and the derived events are passed as STORM tuples to the next stage in the SPEEDD topology processing (see D6.1 in[1]). The distributed environment is part of the work in the SPEEDD prototype in WP6 and out of the scope of this report.

4.2.1.2.1 Producers definition

A producer introduces events from the outside world to the event-processing network. A producer definition includes the following:

- Type – the adapter type this producer is using to push or pull events into the EPN. The supported types are File, JMS, Rest, and custom adapter. Each adapter type has built-in parameters and other parameters can be added. Each parameter has a name and value. The adapter types and their parameters include the following:
  - File – using this adapter type, the producer's events would be read from a given file. A file producer has the following additional built-in parameters:
    - filename – full path file name.
  - Timed – timed file adapter. The events from the file will be injected not at a constant rate, but based on the relative difference of *OccurenceTime* attribute value of the event as

specified in the event row in the file from start of injection. The timed adapter has the same properties as file adapter

- Rest –this adapter type is a REST client that GETs events from an external REST service periodically. A Rest type producer has the following additional built-in parameters:
  - URL – the fully qualified URL of the REST service for event pull operation using a GET method.
  - ContentType – can be "text/plain", "application/xml", or "application/json". This is defined by the REST service.
  - PollingMode – whether the web service returns a single instance or batch of event instances.

  Note: PROTON includes a REST service that provides the ability to push (notify) events to the engine, see CEP open specification document
- Custom – using this adapter type, the producer's event would be read using a custom mechanism defined by the user. In this case, a new type of adapter needs to be added to the adapter framework, as described in the PROTON programmer's guide.

Additional parameters common to all producer types are:

- pollingInterval – the time to wait between two consecutive accesses to the source to pull events.
- sendingDelay – a delay between sending events into the EPN (mainly for demo purposes).
- formatter – the format of the input events (the supported formatters are tag, csv, and json).
- delimiter – the delimiter used to separate between different event attributes.
  - 'tag' type formatter – the delimiter defines the separator between key-value pairs. Default is ";".
  - 'csv' type formatter – the delimiter defines the separator between values. Default is ",".
- tagDataSeparator – for a tag type formatter, the separator between event attribute name and its value. Default is "=".
- csvEventType – for csv type formatter, the name of the event that is received from the producer.
- csvAttributeNames – for csv type formatter, since CSV files only list values, and not keys, of event's attributes, csvAttributeNames are used as keys. csvAttributeNames is a comma-separated string of the attributes in the order they appear in the CSV file (e.g Attribute1, Attribute2, Attribute3…). When the CSV file is read, it will link the first value to the first attribute in csvAttributeNames, and so on.
- dateFormatter - the default date format is dd/MM/YYYY-HH:mm:ss. If you would like to use a different format for your input events, you have to specify a date formatter (e.g., dd.MM.yyyy G 'at' HH:mm:ss z).

For custom adapters, additional required parameters can be added. Each such parameter has a name and a value.

4.2.1.2.2 Consumers definition

A consumer consumes events generated by the EPN and sends them to the outside world. A consumer definition includes the following:

- Type – the adapter type that is used to push or pull events from the EPN. The supported types are File, JMS, Rest, and custom adapter. Each adapter type has built-in parameters and other parameters can be added. Each parameter has a name and value. The adapter types and their parameters:
    - File – using this adapter type, the consumer's events would be written to a given file. A file consumer has the following additional built-in parameters:
        - filename – full path file name.
    - Rest –this adapter type is a REST client that POSTs events to an external REST service upon detection of derived events. A Rest type consumer has the following additional built-in parameters:
        - URL – the fully qualified URL of the REST service for event push operation using the POST method.
        - ContentType – can be "text/plain", "application/xml", or "application/json". This is defined by the REST service.
        - AuthToken – an optional parameter, that when set, is added as an X-Auth-Token HTTP header of the request.
    - Custom – using this adapter type, the consumer's events would be written using a custom mechanism defined by the user. In this case, a new type of adapter needs to be added to the adapter framework, as described in the PROTON programmer's guide.

Additional parameters common to all producer types are:

- formatter – the format of the input events (the supported formatters are tag, csv, and json).
- delimiter – the delimiter used to separate between different event attributes.
    - 'tag' type formatter – the delimiter defines the separator between key-value pairs. Default is ";".
    - 'csv' type formatter – the delimiter defines the separator between values. Default is ",".
- tagDataSeparator – for a tag type formatter, the separator between event attribute name and its value. Default is "=".
- csvEventType – for csv type formatter, the name of the event that is received from the producer.
- csvAttributeNames – for csv type formatter, since CSV files only list values, and not keys, of event's attributes, csvAttributeNames are used as keys. csvAttributeNames is a comma-separated string of the attributes in the order they appear in the CSV file (e.g Attribute1, Attribute2, Attribute3…). When the CSV file is read, it will link the first value to the first attribute in csvAttributeNames, and so on.

- dateFormatter - the default date format is dd/MM/YYYY-HH:mm:ss. If you would like to use a different format for your output events, you have to specify a date formatter (e.g., dd.MM.yyyy G 'at' HH:mm:ss z).

For custom adapters, additional required parameters can be added. Each such parameter has a name and a value.

In the credit card CEP application we run PROTON stand-alone version and used a CSV file for input and a text file for output (as per Feedzai's request). For producer and consumer definitions and types of adapters refer to PROTON's User Guide[5]. Below is a snippet from the JSON definition file related to the consumers and producers for the credit card fraud detection application.

```
"consumers": [
    {
     "name": "FeedzaiConsumer",
     "createdDate": "Wed Aug 06 2014",
     "type": "File",
     "properties": [
      {
       "name": "filename",
       "value": "D:\\EP\\Projects\\EU\\SPEEDD\\usecases\\Feedzai\\exampleOutput.txt"
      },
      {
       "name": "formatter",
       "value": "tag"
      },
      {
       "name": "delimiter",
       "value": ";"
      },
      {
       "name": "tagDataSeparator",
       "value": "="
      },
      {
       "name": "SendingDelay",
       "value": "1000"
      },
      {
       "name": "dateFormat",
       "value": "dd/MM/yyyy-HH:mm:ss"
      }
     ],
     "events": [
      {
       "name": "Transaction"
```

[5] https://github.com/ishkin/Proton/tree/master/documentation

```
      },
      {
       "name": "TrendAfterCount"
      },
      {
       "name": "CountFraud"
      },
      {
       "name": "Fraud"
      },
      {
       "name": "SequenceFraud"
      },
       {
       "name": "IncreasingAmounts"
      },
       {
       "name": "IncreasingAmountsCardIndication"
      },
       {
       "name": "FraudAtATM"
      }
    ]
  }
],
"producers": [
 {
  "name": "FeedzaiProducer",
  "createdDate": "Wed Aug 06 2014",
  "type": "File",
  "properties": [
   {
    "name": "filename",
    "value": "D:\\EP\\Projects\\EU\\SPEEDD\\usecases\\Feedzai\\exampleInputUncertain.txt"
   },
   {
    "name": "pollingInterval",
    "value": "1000"
   },
   {
    "name": "sendingDelay",
    "value": "1000"
   },
   {
    "name": "formatter",
    "value": "tag"
   },
```

```
    {
     "name": "delimiter",
     "value": ";"
    },
    {
     "name": "tagDataSeparator",
     "value": "="
    },
    {
     "name": "dateFormat",
     "value": "dd/MM/yyyy-HH:mm:ss"
    }
   ],
   "events": []
  }
]
```


4.2.2 Evaluation results

As explained above we needed to test our application by Feedzai's people at their premises twice: once including uncertainty (the "uncertain case") and once without the inclusion of uncertainty aspects (the "certain case") as detailed below.

As a first step in order to validate the implemented EPN and before running the application with large amounts of historical data, we tested the application using generated data and run the input data twice: with uncertainty and without.

4.2.2.1 Sample data

For the first step, i.e. generated data by us, we used several tens of input events. The amount enabled us to test the different patterns and the effectiveness of using uncertainty in the derived events for the CP case.

The real data stored ~1 million transactions, with 122 transactions flagged as fraudulent. Most of these 122 transactions were result of strikes that covered only two transactions, and therefore, could not be detected by our *FlashAttack* pattern.

4.2.2.2 Results

In the first step, we compared the derived events for the two cases (certain and uncertain) for each of the patterns. We show below two examples:

IncreasingAmounts – In this pattern we start firing situations after 4 times, with an increase certainty value as can be seen by the small snippet below taken from the output file.

{"Name":"**IncreasingAmounts**","**TrendCount**":"**4**","transaction_ids":"[f13b2a6b0fdc64a7391882bc0ee4 0660, ert564b0fdc64a7391882bc0ee456t33, 345a6b0fdc64a73913452bc0ee40345, 345a6b0fdc64a73346y82bc0345y3455]","Certainty":"**0.5998883688639982**","card_pan":"8896141"}

{"Name":"**IncreasingAmounts**",**TrendCount":"5"**,"transaction_ids":"[f13b2a6b0fdc64a7391882bc0ee40660, ert564b0fdc64a7391882bc0ee456t33, 345a6b0fdc64a73913452bc0ee40345, 345a6b0fdc64a73346y82bc0345y3455,456rtyr564a7391882bc0ee45648563dg]","Certainty":"**0.7498851783023105**","card_pan":"8896141"}

"Name":"**IncreasingAmounts**","**TrendCount":"6"**,"transaction_ids":"[f13b2a6b0fdc64a7391882bc0ee40660, ert564b0fdc64a7391882bc0ee456t33, 345a6b0fdc64a73913452bc0ee40345, 345a6b0fdc64a73346y82bc0345y3455, 456rtyr564a7391882bc0ee45648563dg, tyrtyhdc64a7391882bc0eertyndertssh]","Cost":"0.0","**Certainty":"0.8570498356842731**","card_pan":"8896141"}

{"Name":"**IncreasingAmounts**","**TrendCount":"7"**,"transaction_ids":"[f13b2a6b0fdc64a7391882bc0ee40660, ert564b0fdc64a7391882bc0ee456t33, 345a6b0fdc64a73913452bc0ee40345, 345a6b0fdc64a73346y82bc0345y3455, 456rtyr564a7391882bc0ee45648563dg, tyrtyhdc64a7391882bc0eertyndertssh, rtyhrtymme5c64a7391882bc0etybwe5]","**Certainty":"0.923012520878032**","card_pan":"8896141"}

In the certainty version we derive a single situation only after the seventh time (see Table 3). As can be seen, even in this simple case, if we had flagged the forth transaction already as in the uncertain case, then we could have "saved" 3 fraudulent transactions from taking place. Based on the fact that this pattern indicates increasing amounts, the later the transactions are in time, the more expensive.

MultipleATMWithdrawals - In this pattern we start firing situations after 3 times (see Table 3), with an increase certainty value as can be seen by the small snippet below taken from the output file.

{"Name":"**MultipleATMWithdrawals**","**Certainty":"0.7001474286095372"**,"terminal_id":"1948456145","transaction_ids":"[rtyhrtymme5c64a7391882bc0etybwe5, 456hs7ds0fdc64a7391882bc0ee34sg35, frtykj56b0fdc64a7391882bc0ee457srs4]",**,"TransactionsCount":"3.0**"}

{"Name":"**MultipleATMWithdrawals**","**Certainty":"0.9000697663968664"**,"terminal_id":"1948456145", "transaction_ids":"[rtyhrtymme5c64a7391882bc0etybwe5, 456hs7ds0fdc64a7391882bc0ee34sg35, frtykj56b0fdc64a7391882bc0ee457srs4, 2e526cef7eb66823c1ccadd69135d9d2]","**TransactionsCount":"4.0**"}

{"Name":"**MultipleATMWithdrawals**","**Certainty":"0.9720230891240956"**,"terminal_id":"1948456145", "transaction_ids":"[rtyhrtymme5c64a7391882bc0etybwe5, 456hs7ds0fdc64a7391882bc0ee34sg35, frtykj56b0fdc64a7391882bc0ee457srs4, 2e526cef7eb66823c1ccadd69135d9d2, 2e526cef7eb66823c1ccadd69134567]","**TransactionsCount":"5.0**"}

While in the certain version we fire a situation only at the fifth time, when we could potentially flag the third transaction for the same credit card and prevent for two transactions from taking place.

This first step enabled us to validate the patterns and already indicated the large potential latent in the inclusion of uncertainty aspects in the credit card fraud detection use case. Naturally, the sooner we flag

a fraudulent transaction as such, the largest the savings are. On the other hand, tagging too soon can cause to false positive results.

After achieving these first results we were confident in moving to the next step and testing the application with large amounts of real data.

When running the "uncertain" application on the same real data sample, the application fired 308 situations (uncertain fraud events) belonging to 284 different cards for the two patterns: *IncreasingAmounts* and *FlashAttack*, and one terminal for *MultipleMaxATMWithdrawals* distributed as follows (Table 4):

Table 4: Number of transactions per detected pattern in the uncertain CP credit card use case

#transactions in the matching set	FlashAttack	IncreasingAmounts	MultipleMaxATMWithdrawals
3	157		1
4	69	12	1
5	31	2	1
6	10		
7	11		
8	7		
9	5		
10	1		
total	291	14	3

Note that all the cards belonging to *IncreasingAmount* also were detected by *FlashAttack*. That means that the transactions were made in a relatively short period of time and at the same time their amounts were in an increasing order. In addition we fire an *IncreasingAmount* derived event in an immediate and reuse modes, meaning that the derivations with 5 transactions also are derivations with 4 transactions. Furthermore, 7 out of the *FlashAttack* detections were fired twice for the same card_pan. This means that the same credit card caused the pattern to be matched two times in two different temporal windows. This gives a total of 284 different credit card numbers (291-7=284).

Some of the more "interesting" situations in which more than one patterns was satisfied along with a relatively number of transactions are (X, Y, Z, and U denote credit card numbers as we don't have the real values) :

- For credit card X: FlashAttack : count = 3; IncreasingAmounts: count = 4; IncreasingAmounts: count = 5
- For credit card Y: IncreasingAmounts :count = 4; FlashAttack : count = 8
- For credit card Z: FlashAttack: count = 5; FlashAttack:count = 9 (two different time windows)
- For credit card U: FlaskAttack: count = 8; IncreasingAmounts: count = 4; IncreasingAmounts: count = 5

When running the "certain" application on real data in Feedzai, the application detected 35 situations as (certain) fraud, 34 for *FlashAttack* and 1 for *MultipleMaxATMWithdrawals.* This can be also be checked by the uncertain case (Table 4) as *FlashAttack* is fired for the certain case only after 6 transactions (34=10+11+7+5+1), for *IncreasingAmounts* after 7 transactions (none) and *MultipleMaxATMWithdrawals* after 5 (only 1 transaction).

A comparison between our findings and the annotated data reveals no overlapping, that is we detected situations not flagged in the data and vice versa. Several reasons to this:

- Our patterns detect situations with more than three transactions in the pattern, while many of the transactions flagged belong to one or two transactions only.
- We only detect some of the possible patterns while there are others that we haven't implemented yet. On the other hand, from those implemented we can wonder why the transactions are not flagged as fraud. Possible reasons: the temporal windows and the thresholds are not fined tuned enough. It might also be that we were able to detect a few transactions that slipped away from Feedzai's system. As we don't have the data our conclusions are inconclusive.
- Another important factor is that we don't perform enrichment of the data. Adding information such as history of customer provides additional and valuable information at Feedzai.

4.2.2.2.1 Summary of results

Our results show the potential latent in including uncertainty aspects in an event driven application in the domain of credit card fraud. The fact that we can alert ahead of time has a significant financial impact as we can avoid performing "future fraud transactions". Although we were able to detect 284 different credit cards as potential fraud at different levels of confidence, these were not validated in the annotated data. Our results are not conclusive and further analysis need to be performed. This will be our main focus during year three of the project.

5 Event processing application for the traffic management use case

In this section we describe the work carried out in the fraud detection use case including the design of the event processing network, the implementation of the application, and its evaluation using real-data.

5.1 Design of second EPN for the traffic management use case

A second version of the EPN for the traffic management use case that includes improvements and insights gained was devised during the second year of the project. The resulting EPN consists of eight EPAs is shown in Figure 23 and detailed in the following Sections. For the sake of simplicity we only show the EPAs and the events flow in the network. Dotted lines represent events, other than input events, that are either initiators (in yellow) or terminators (in red) of a context. The PROTON JSON definitions

file that comprises this EPN is provided as part of the software deliverable that accompanies this report. In the current EPN we fire situations in the following cases:

- A *Congestion* (EPA2) in a specific location is building-up.
- A *ClearCongestion* (EPA3) at a specific location is identified.
- A *PredictedCongestion*, that is, a forecasted congestion is identified at a specific location (EPA4 and EPA5).
- Calculations on sensor readings are emitted to be consumed by the decision making module (EPA6 and EPA7)

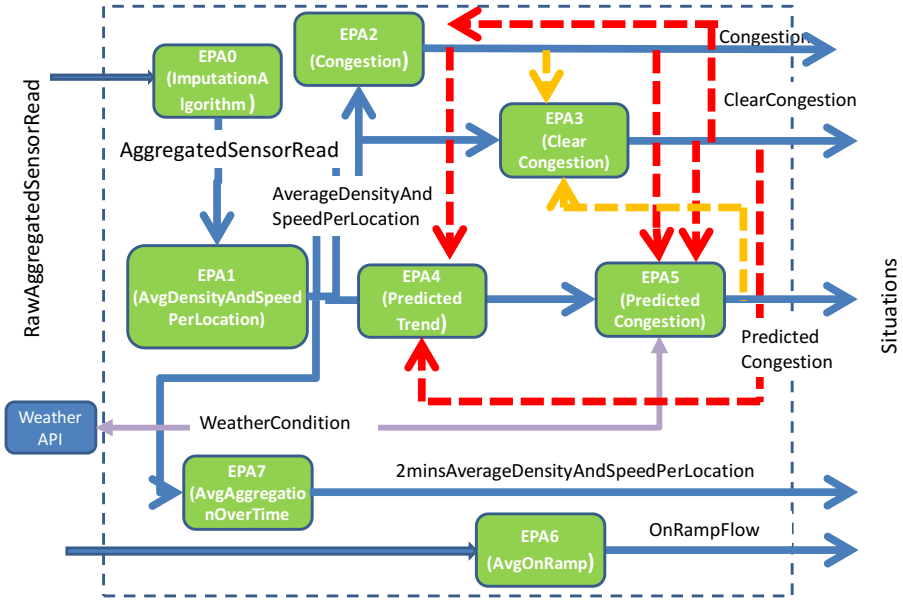

Figure 23: Traffic management use case EPN

Note that EPA0 is an extended EPA (see Section 2.2) whose role is to correct errors in raw sensor readings by embedding the imputation algorithm developed by the CNRS team. In addition, note that for all the detections apart of EPA0 and EPA6, average measurements are taken (provided by EPA1).

5.1.1 Calculations of congestion, clear congestion, and predicted congestion situations

As in our previous report, we differentiate among three situations at a specific location using two parameters: density and speed. A *Congestion* exists if the density in a specific location is above a certain given value (*density_threshold1*) and the speed is below a certain given value (*speed_threshold1*). On the other hand, a congestion is over (*ClearCongestion*), whenever the density is below a certain given value (*density_threshold2*) and the speed is above a certain given value (*speed_threshold2*). We emit a *PredictedCongestion* situation in between the *Congestion* and the *ClearCongestion* thresholds. Note that for the speed we added a new threshold (*speed_threshold3*) in order to narrow the limits for a *PredictedCongestion*, but of course, any value between *speed_threshold1* and *speed_threshold2*, can be selected.

5.1.2 Calculation of density

For the density values we follow the formula given by CNRS and based on [2] and [3] that correlates density between two adjacent locations and the occupancy values at these locations. That is, for any two consecutive locations l_d and l_u forming a section p given the occupancy measurements $o_d(t)$ and $o_u(t)$ at time t, the density can be computed according to the following formula (1) where Γ is a known coefficient. At this stage, since we do all our processing at sensor locations we approximate the formula to density at a location and not between two locations. The coefficients are given by the CNRS team.

(1) $$\rho_p(t) \leftarrow \Gamma_p \frac{o_d(t)+o_u(t)}{2}$$

5.1.3 Weather API

One of the advantages of applying complex event processing is the capability of analyzing and processing events coming from heterogeneous sources. In order to test this, we "combined" the *PredictedCongestion* event with weather conditions. The rationale is as follows: a forecasted event probability/certainty is higher knowing bad weather conditions are present. To this end, we apply the open API given by Weather Underground to receive alerts on current conditions such as fog, rain, snow, and temperature by invoking:

http://api.wunderground.com/api/82f05a9f463df021/alerts/q/FR/grenoble.json

The *Weather API* is synchronously called by the *PredictedCongestion* EPA (see Figure 23) to calculate the certainty of the *PredictedCongestion* situation. See Section 5.1.5.5 below.

5.1.4 Event types

Ten event types have been defined that comprise the event inputs, outputs/derived, and situations (Table 5). For the sake of simplicity we only show the user-defined attributes or the event payload and not the metadata.

Although the names of concepts can be determined freely by the application designer in PROTON, we use some naming conventions for the sake of clarity. We denote event types with capital letters. Built-in/metadata attributes start with a capital letter, as well as payload attributes that hold operators values, while payload attributes start with a lower letter. Table 5 shows the event definitions for the traffic management EPN. Note that the *problem_id* attribute is not part of the raw event payload and is intended for monitoring reasons by the decision and frontend modules of the SPEEDD prototype. At this stage, we assign the *location_id* value to the *problem_id* at the derivation step, but more complex expressions can be applied.

Note that the *RawAggregatedSensorRead* raw event includes more fields or attributes. We defined only the ones required for pattern detection in the EPN implementation. When running PROTON will ignore event attributes not specified in its JSON definition file.

Table 5: Event types for the traffic management use case

Event name	RawAggregatedSensorRead
Payload	location, lane, occupancy, vehicles, average_speed
Event name	AggregatedSensorRead
Payload	location, lane, occupancy, vehicles, average_speed
Event name	Congestion
Payload	location, average_density, problem_id
Event name	PredictedCongestion
Payload	location, average_density, problem_id
Event name	ClearCongestion
Payload	location, problem_id
Event name	OnRampFlow
Payload	location, average_flow, average_speed, average_density
Event name	AverageDensityAndSpeedPerLocation
Payload	location, average_flow, average_density, average_ speed
Event name	2minsAverageDensityAndSpeedPerLocation
Payload	location, average_flow, average_speed, average_density
Event name	PredictedTrend
Payload	location, average_density, problem_id
Event name	WeatherCondition
Payload	Current weather conditions, e.g., temperature, humidity and wind, as provided bythe API http://api.wunderground.com/api/82f05a9f463df021/conditions/q/FR/grenoble.json

5.1.5 Event processing agents

Henceforth, we describe the EPAs in the following order: Event name; motivation; event recognition process (following Figure 2); contexts along with temporal context policy; and pattern policies.

In the event recognition process we only show the steps that take place in the specific EPA, while the others are greyed. For the *filtering step* we show the filtering expression; for the *matching step* we denote the pattern variables; and for the *derivation step* we denote the values assignment and calculations. Please note that for the sake of simplicity we only show the assignments that are not copy of values (all other derived event attributes values are copied from the input events). For attributes, we just denote their names without the prefix of 'attribute_name.'

EPA0: ImputationAlgorithm is an *extended EPA* in PROTON in the sense that it is not one of the built-in EPAs or operators existing in the tool but it was written to perform the calculation of the imputation algorithm given by the CNRS team. PROTON enables extending the list of its operators by providing specific hooks in the run-time and metadata for that purpose. The aim of this EPA is to convert the errors in inputs (the "-1" in the event input data) to meaningful values. For details on the algorithm refer to [4].

Note that EPAs 6, and 7 remained the same as in our previous version, as these EPAs compute averages consumed by external modules to PROTON. We provide their description below for the sake of clarity and completeness.

5.1.5.1 EPA1: AvgDensityAndSpeedPerLocation

Motivation: This EPA calculates averages of speed and occupancy to derive an average density over all the lanes in a certain location except for on-ramp lanes, which are treated differently.

Event recognition process:

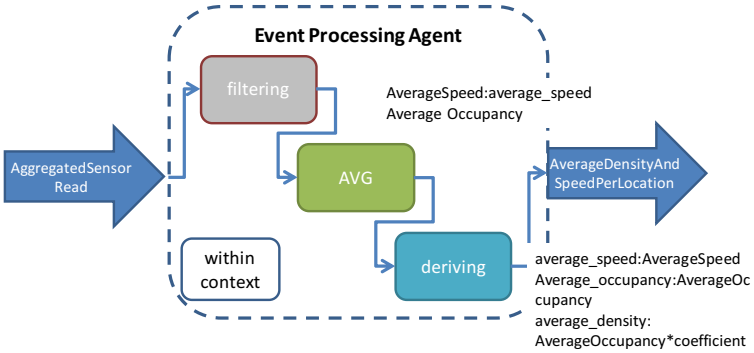

Figure 24: Event recognition process for AvgDensityAndSpeedPerLocation EPA

AverageOccupancy and *AverageFlow* are computed variables of the AVG pattern.

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	SINGLE	FIRST	CONSUME

Context:

Segmentation: by location_id

Initiator policy: IGNORE

Meaning: For each *AggregatedSensorRead* at each location, there is a derived event for the average density and speed. As there is one event every 15 sec, the initiator policy doesn't play a role in this case.

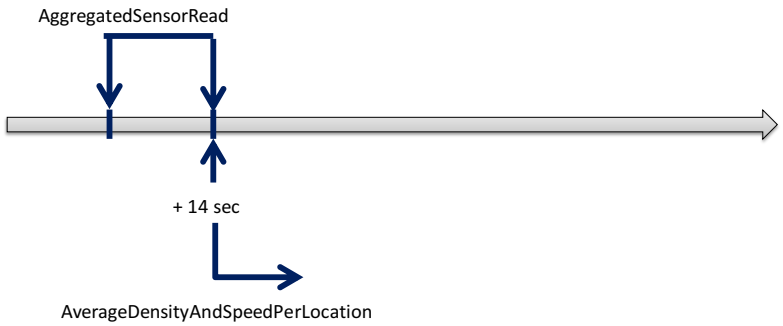

Figure 25: Context for AvgDensityAndSpeedPerLocation EPA

5.1.5.2 EPA2: Congestion

Motivation: To derive a congestion alert whenever the average density and speed are above and under specific given thresholds (labeled by 1) by at least 15 events (in order to reduce fluctuations we check that the situation remains constant for at least 15 times). In this case the derived event has a certainty value of 1, since the congestion detected is already taking place. We apply the values of density_threshold1:0.6 and speed_threshold1:30, given by the simulator expert in CNRS.

Event recognition process:

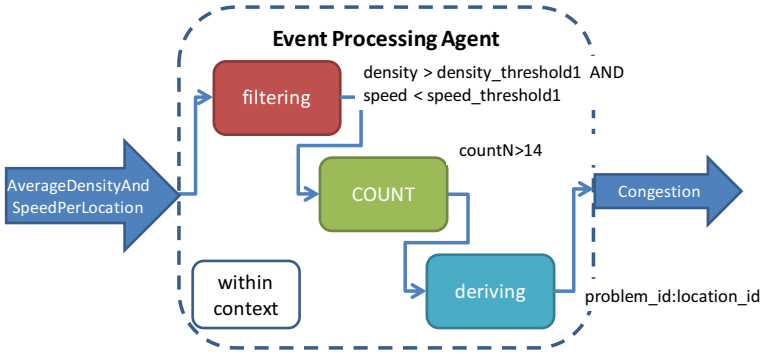

Figure 26: Event recognition process for Congestion EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	SINGLE	FIRST	CONSUME

Context:

Segmentation: by location_id

Initiator policy: IGNORE

Meaning: We open a single context for location in order to detect an actual congestion which is immediate fired once the pattern is matched. The context is closed with the *ClearCongestion* event, i.e., the congestion passed, or after 5 min.

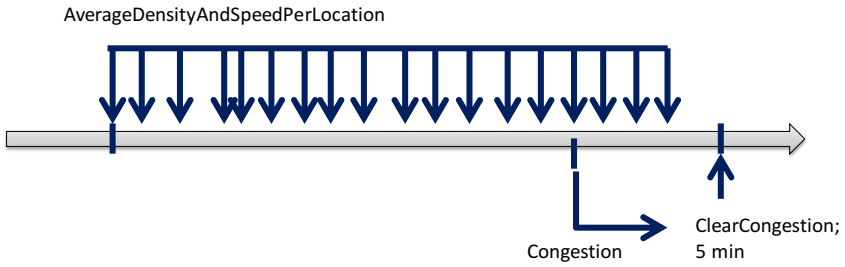

Figure 27: Event recognition process for Congestion EPA

5.1.5.3 EPA3: ClearCongestion

Motivation: A derived event is emitted whenever the flow is perceived as "normal", meaning the density and speed thresholds are in the "normal range" by at least 15 events (in order to reduce fluctuations we check that the situation remains constant for at least 15 times). In this case the derived event has a certainty value of 1. We apply the values of density_threshold2:0.50 and speed_threshold2:80, given by the simulator expert in CNRS.

Event recognition process:

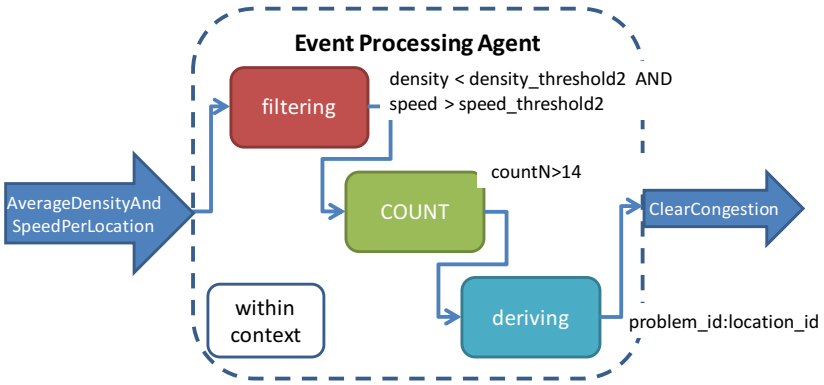

Figure 28: Event recognition process for ClearCongestion EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	SINGLE	FIRST	CONSUME

Context:

Segmentation: by location_id

Initiator policy: IGNORE

Meaning: We open a single context for a location in order to detect when an occurring congestion or predicted congestion goes away. To this end, the context is opened with either the *Congestion* or *PredictedCongestion* events (the first that comes) and is closed after 5 minutes.

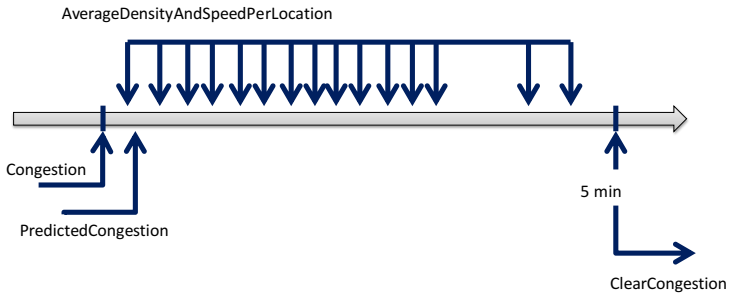

Figure 29: Context for ClearCongestion EPA

5.1.5.4 EPA4: PredictedTrend

Motivation: The CEP run-time engine will derive a *PredictedTrend* event (input to the *PredictedCongestion* EPA) whenever it detects an increase in the density values of at least 5 consecutive input events. In other words, we perceive that a build-up is taking place possible indicating a congestion in the near future. The idea is that we only derive an event if the build-up hasn't reached either a congested or a clear congested state and therefore we added this condition in the derivation step. We check whether both density and speed are still in the normal range for the last event in the matching set. In the case of speed we used the value of 40 for our tests for speed_threshold3, to avoid reaching a congested state.

We use the *Sigmoid* function for the probability of the derived that receives three parameters and four coefficients and returns 1/ (1 + e^-(a + b1x1+b2x2+b3x3)), where:

- X1 – denotes the gradient between the difference of density of the last and first events in the matching set and the elapsed time between the two.
- X2 – denotes the difference between the density threshold for congestion (0.95) and the density of the last event in the matching set.
- X3 – *trend.count* (number of events in the matching set)

The rationale is that we would like to give more probability to a derived event which is closer to the congestion *density_threshold*, the ratio is higher (the build-up of congestion occurs faster), and the number of events is larger. As in the fraud detection use case, the values for the coefficients were calculated using different combinations for targeted probabilities and *X,* and solving the equations (see Section 5.2.1.1).

In addition, we also increase by 0.1 the probability of a predicted congestion when the temporal window occurs during rush hours but not weekends as a further fine tuning.

Event recognition process:

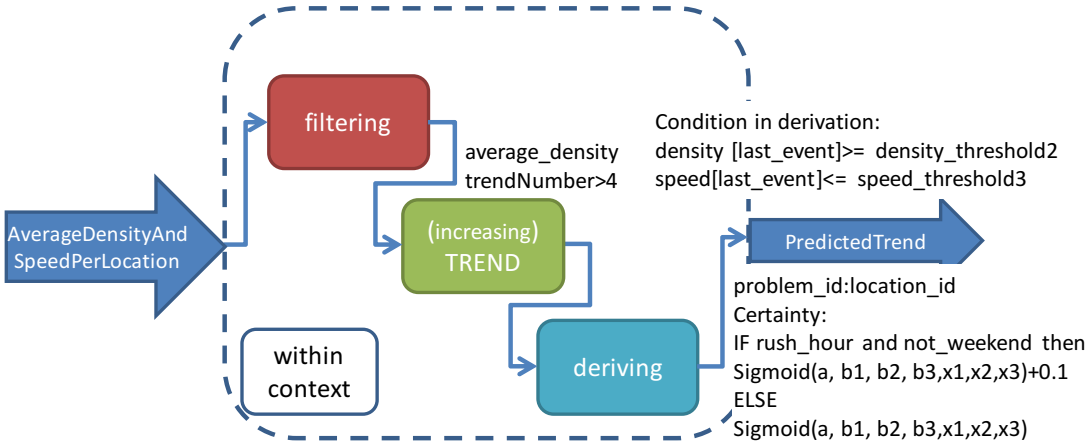

Figure 30: Event recognition process for PredictedTrend EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by location_id

Initiator policy: IGNORE

Meaning: We open a single context for a location in order to detect an increasing TREND pattern. To this end, the context is opened with the first input event that comes and is closed when either a *Congestion* or *ClearCongestion* is detected for the same location. As we use the IMMEDIATE and UNRESTRICTED policies, we derive a *PredictedTrend* event for each TREND encountered; with increasing *Certainty* value if the buildup continues (density continues to rise and speed continues to decrease).

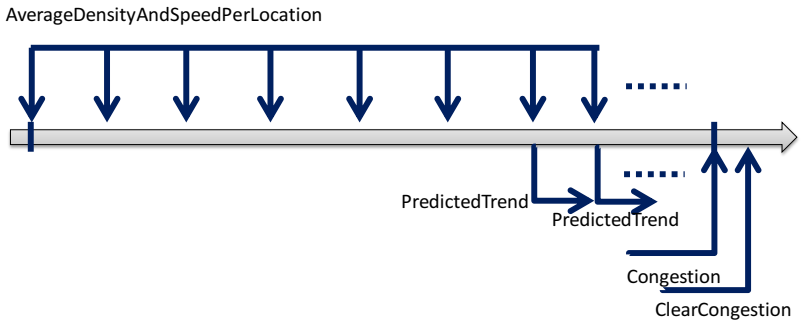

Figure 31: Context for PredictedTrend EPA

5.1.5.5 EPA5: PredictedCongestion

Motivation: We combine the *PredictedTrend* event with the *WeatherCondition* event. If the latter denotes a bad condition (i.e. fog, rain, or snow) then, we increase a little bit the probability of the *PredictedCongestion* event (+0.1).

Event recognition process:

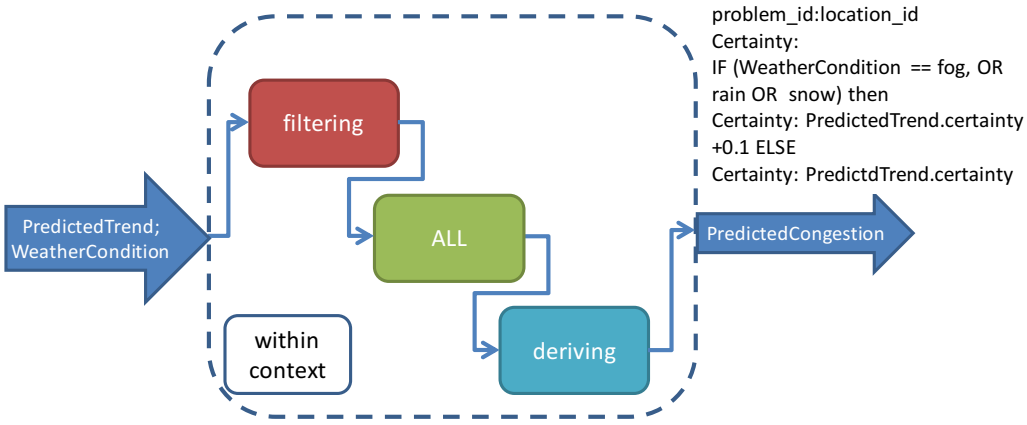

Figure 32: Event recognition process for PredictedCongestion EPA

Pattern policies:

We denote *WeatherCondition* as e1 and *PredictedTrend* as e2.

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	e1: FIRST e2: OVERRIDE	e1: REUSE e2: CONSUME

Context:

Segmentation: by location_id

Initiator policy: IGNORE

Meaning: The context is opened with the first *PredictedTrend* event that is emitted from EPA4 which causes the synchronous call of the *WeatherCondition* event. As in the case of WP4, the temporal context is closed when either *Congestion* or *ClearCongestion* is detected for the same location.

As we apply the IMMEDIATE and UNRESTRICTED policies, we can derive more than one event. We apply the FIRST and REUSE policies for *WeatherCondition*, meaning, we re-use the event that is invoked for each of the ALL combinations with the *PredictedTrend* input event. In the latter, we apply the OVERRIDE and CONSUME policies; therefore we use a new input event for each combination of the ALL pattern.

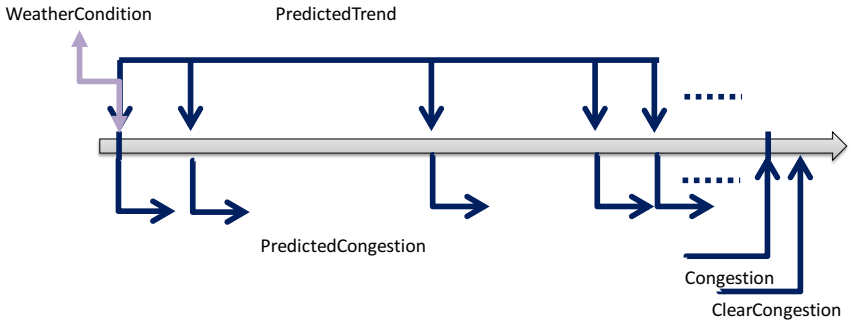

Figure 33: Context for PredictedCongestion EPA

5.1.5.6 EPA6: AvgOnRamp

Motivation: To derive average values of on-ramp lanes every two minutes for the decision module.

Event recognition process:

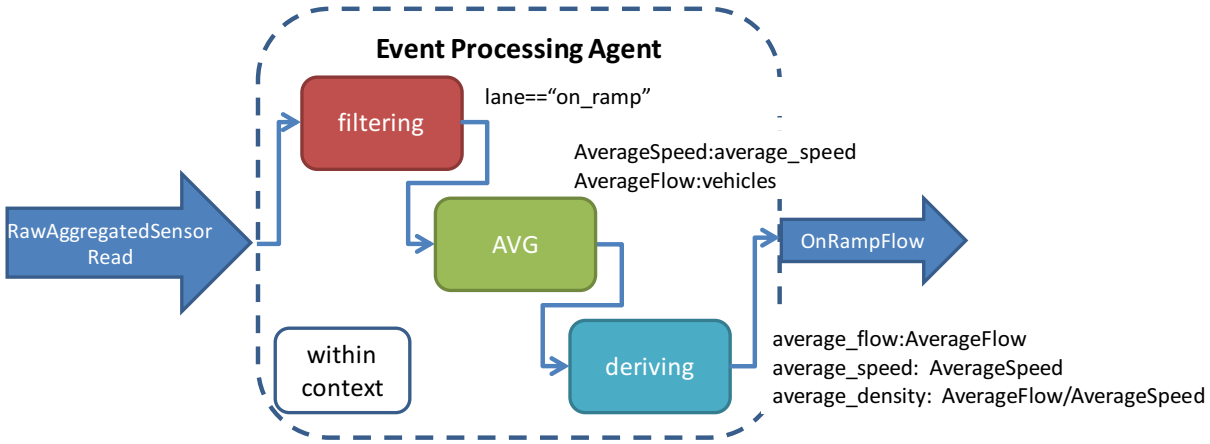

Figure 34: Event recognition process for AvgOnRamp EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	SINGLE	FIRST	CONSUME

Context:

Segmentation: by location_id

Initiator policy: ADD

Meaning: In each location, for each input event (sliding/overlapping temporal windows) we open a temporal context and perform average calculations. The temporal contexts are closed after two minutes.

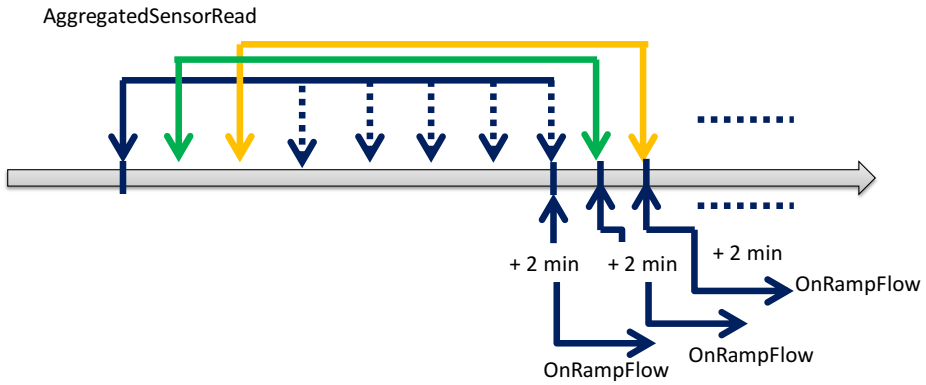

Figure 35: Event recognition process for AvgOnRamp EPA

5.1.5.7 EPA7: AvgAggregationOverTime

Motivation: To derive average values for all lanes every two minutes for the decision module.

Event recognition process:

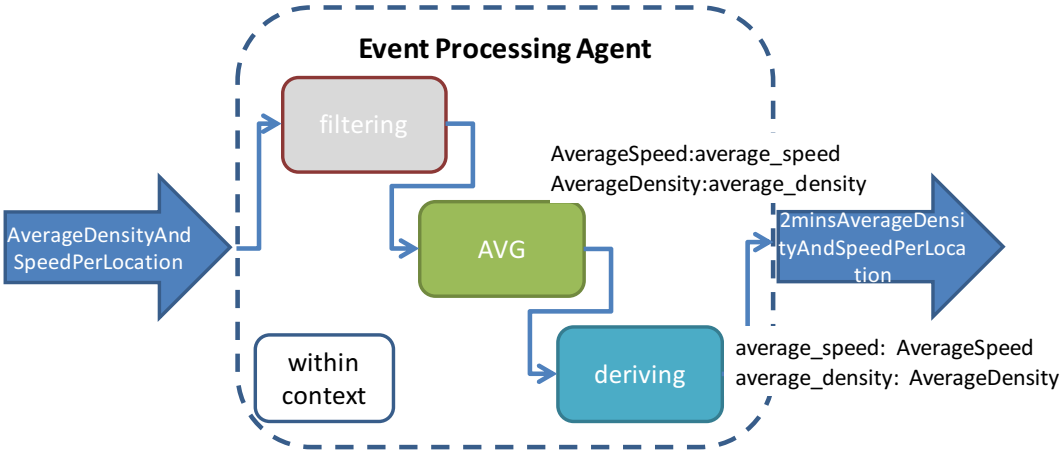

Figure 36: Event recognition process for AvgAggregationOverTime EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
DEFERRED	SINGLE	FIRST	CONSUME

Context:

Segmentation: by location_id

Initiator policy: ADD

Meaning: In each location, for each input event (sliding/overlapping temporal windows) we open a temporal context and perform average calculations. The temporal contexts are closed after two minutes.

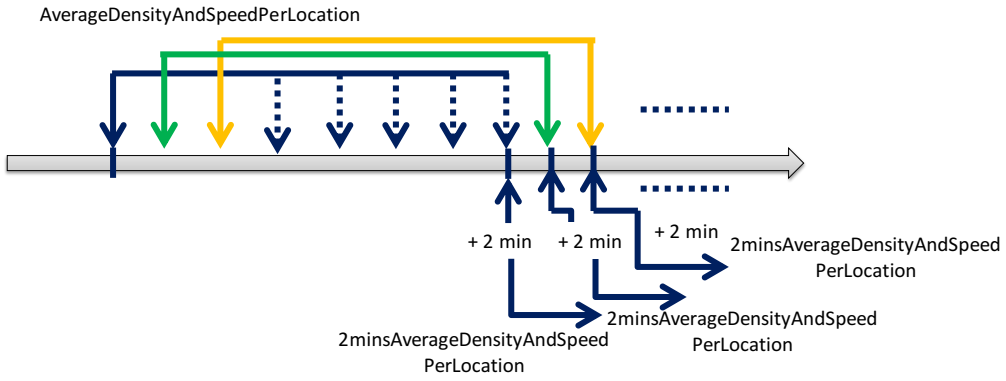

Figure 37: Event recognition process for AvgAggregationOverTime EPA

5.2 Implementation and evaluation of second EPN for the traffic management use case

During the second year of the project we had two main goals with regards to the traffic use case:

- First, to compare the situations emitted from the CEP application with actual real congestions. In order to do so, we needed annotated data, that is, timestamps for events during an elapsed time window that caused a sudden decrease in flow/sudden increase in density. The evaluation should answer the question: Can we forecast a congestion before it actually happens? In other words, is the inclusion of uncertainty aspects and the ability to predict a future event effective?
- Second, we wanted to have one version of the CEP application that can be tested in a stand-alone mode and also integrated into the SPEEDD prototype in closed-loop.

In order to meet these two requirements, we decided to perform our tests using synthetic data generated by the Aimsun simulator at Grenoble and provided by the NCRS team. For details on the simulator refer to D8.2 "First Version of Micro-Simulator"[6].

Pros

- Compatible with the rest of the project that uses the Aimsun data sample for testing purposes.
- Aimsun can provide annotated data of incidents we can compare our results against.
- The same sample data can be used for evaluation of the component alone and for the integrated prototype.
- Simplification of calculations – The synthetic data doesn't produce erroneous inputs and therefore there is no need to apply the imputation algorithm of NCSR at this stage. In addition, we don't need to approximate density values, as these are provided by the synthetic data (input to the CEP component)

[6] Available at: file:///D:/SPEEDD/WP3%20event%20processing%20under%20uncertainty/D3.2/SPEEDD-D8-2.pdf

Cons

- We can only run event patterns that apply to the highway route, as the simulator at this stage generates events for the high way and only a small portion of the downtown (the full mapping of the downtown will be accomplished in D8.4 due at M24).
- The Aimsun simulator cannot connect to an external API; therefore we cannot connect to the weather API.

5.2.1 Implemented EPN

Utilizing data generated by Aimsun results in a simplified EPN shown in Figure 38. EPA0 (imputation algorithm), EPA1 (AvgDensityAndSpeedPerLocation), and EPA5 (combining predicted events with weather conditions) are excluded as they are not required when using simulated data.

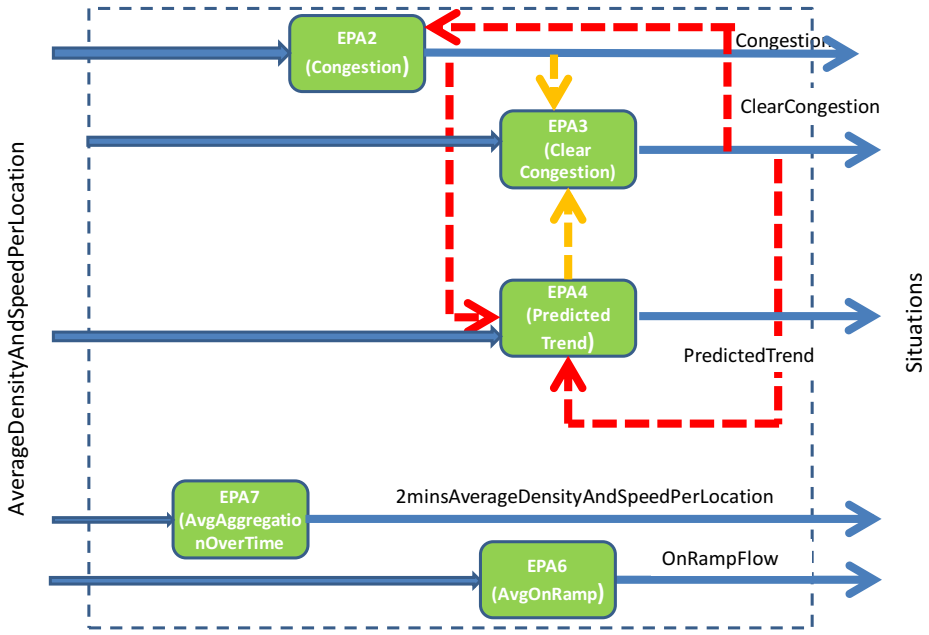

Figure 38: Implemented EPN for the traffic management use case

5.2.1.1 *Values of operator operands and coefficients of the Sigmoid function*

As aforementioned, two types of tests have been conducted: the "uncertain case" and the "certain case". For the "uncertain case", the following values for the *Sigmoid* variables for the *PredictedTrend* EPA have been chosen (Table 6). The values have been selected by assigning probabilities for the two extreme conditions of "congestion" (close to 1) and "clear congestion" (close to 0) and looking at the simulated data.

Table 6: Variable values calculation for Sigmoid function for the PredictedTrend EPA

x1	X2	X3	probability
0.0046	0.6576	5	0.269
0.0040	0.697	6	0.5

0.00386	0.727	7	0.731
0.0024	0.787	9	0.95

The coefficients that match are:

a: -6.667; b1: 70.086; b2: 0.447; and b3: 1.010

In the certain case, *PredictedTrend* EPA degenerates into EPA2 (Congestion) and is no longer part of the EPN.

5.2.1.2 Input and output adapters/Producers and consumers

As in the credit card fraud detection application (see Section 4.2.1.2), we also used a CSV file for input and a text file for output in the traffic use case as defined in the snippet of the JSON file below:

```
"consumers":
  [
        {
    "name": "TrafficConsumer",
    "createdDate": "Sun Aug 31 2014",
    "type": "File",
    "properties": [
     {
      "name": "filename",
      "value": "D:\\EP\\Projects\\EU\\SPEEDD\\usecases\\CNRS\\exampleOutput.txt"
     },
     {
      "name": "formatter",
      "value": "tag"
     },
     {
      "name": "delimiter",
      "value": ";"
     },
     {
      "name": "tagDataSeparator",
      "value": "="
     },
     {
      "name": "SendingDelay",
      "value": "1000"
     },
     {
      "name": "dateFormat",
      "value": "dd/MM/yyyy-HH:mm:ss"
     }
    ],
```

```
    "events": [
     {
      "name": "PredictedTrend"
     },
     {
      "name": "AverageDensityAndSpeedPerLocation"
     },
     {
      "name": "Congestion"
     },
     {
      "name": "ClearCongestion"
     }
    ],
    "actions": []
   }
  ],
  "producers":
  [
            {
    "name": "TrafficProducer",
    "createdDate": "Sun Aug 31 2014",
    "type": "File",
    "properties": [
     {
      "name": "filename",
      "value": "D:\\EP\\Projects\\EU\\SPEEDD\\usecases\\CNRS\\simulatorData.csv"
     },
     {
      "name": "pollingInterval",
      "value": "500"
     },
     {
      "name": "sendingDelay",
      "value": "1000"
     },
{
      "name": "formatter",
      "value": "csv"
     },
     {
      "name": "delimiter",
      "value": ","
     },
     {
      "name": "tagDataSeparator",
      "value": "="
```

```
        },
        {
          "name": "csvEventType",
          "value": "AverageDensityAndSpeedPerLocation"
        },
        {
          "name": "csvAttributeNames",
          "value":
"did,location,sid,ent,average_flow,average_speed,average_occupancy,average_density,timestamp,lane
"
        },
        {
          "name": "dateFormat",
          "value": "dd/MM/yyyy-HH:mm:ss"
        }
      ],
      "events": []
    }
]
```


5.2.2 Evaluation results

As aforementioned, we aim at answering the question: is the inclusion of uncertainty aspects and the ability to predict an event effective? The way to address this is to have two applications or EPNs, once including uncertainty aspects and the other one without uncertainty, i.e. deterministic. This is a common approach in CEP engines dealing with uncertainty (see for example in [5]). Therefore, we run our tests twice: once using the EPN as depicted in Figure 38 and once using the same EPN in a deterministic way or without the inclusion of uncertainty. In the latter case the *PredictedTrend* EPA degenerates into EPA2 (Congestion) and is no longer part of the EPN. In other words, in the deterministic case we have two relevant EPAs: EPA2 (*Congestion*) and EPA3 (*ClearCongestion*).

5.2.2.1 Sample data

Data encompassing 3 hours (from 4PM-7PM) has been simulated in Aimsun. A stream is received every 15 seconds as in the real physical sensors. The data from the Aimsun includes the following attributes:

did - Replication ID for random seed

oid - Detector ID

sid - Vehicle type(0 = for all vehicles, 1 = Car , 2 = Truck)

ent - Time interval, from 1 to N, where N is the number of time intervals, and 0 with the aggregation of all the intervals. As the data covers 3 hours, we have a total of 10,800 sec. Having intervals of 15 sec gives 720 intervals for each simulation (10800/15=720).

countveh - Vehicle count

speed - Speed [km/h]

occupancy - Occupancy

density - Density at the specific location [#of vehicles/km]

A total of 10 simulations have been created with random seeds for all the simulations. For each of the simulations a csv file was produced. In addition, another annotated file has been produced containing information about incidents that had been created during the simulations. For each simulation (csv file) an intended incident was created and annotated. An incident causes a rapid build-up of congestion and helps us evaluating our application. Only two out of the 10 incidents occur in the main road, namely in simulation #6 and #10. All other incidents appear in off or on ramps.

The annotated file includes the following fields:

Aimsun Section ID - Where incident occurred

Duration of incident – the time window for the incident [hh:mm:ss:]

Start Time – of the incident [hh:mm:ss:]

Length of Effected Area – in [meters]

In all the ten simulations there is a congestion building up in all sensors close to the end of the road around 18:40 that lasts until the simulation end, with no clearing up of the congestion.

5.2.2.1.1 Data preprocessing

Before injecting the sample data into PROTON, the following steps were carried out to conform to our pattern definitions:

- Filtering out duplicated information - In the input file there were 3 rows to express the number of cars of each type that were simulated for the same interval at the same location. We filtered out the rows in which the number of vehicles for a certain vehicle type was 0 and calculated the distribution between cars and trucks based on the number of cars and trucks values.
- Normalizing the *density* values – In our pattern and Sigmoid function we used normalized values (a bounded value is desirable to know "how far" we are from upper and lower values, i.e. congestion and clear congestion). Therefore, the simulated values were transformed based on the number of cars and trucks and the length of each type: car (4 meters) and truck (7.5) as used in the simulator.
- Mapping of the *oid* (Detector ID) to the *location_id* of the physical sensors in the highway (see Figure 39) and filtering out other locations.
- Calculation of timestamps of events – based on the intervals given. We also filtered out the "interval 0" which the simulator gave as a summary across all intervals for each detector.

- Replacing values of "-1" in the *speed* attribute. The simulator detects "-1" in speed in two (opposite) situations: no car is detected or a car is standing or blocking. In both cases the "-1" is replaced by the previous *speed* value for the same detector.

Once the sample data was ready, we applied the PETITE utility (see Section 3) and "shrinked" the data and the JSON metadata so that we were able to run 3 hours in 20 minutes.

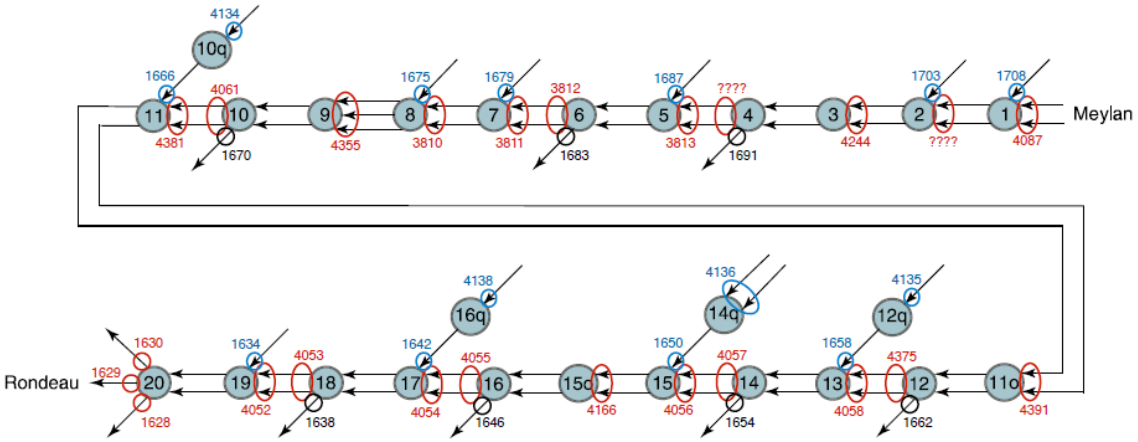

Figure 39: Mapping of the oid (Detector ID) to the location_id of the physical sensors in the highway

5.2.2.2 Results

Out of the 10 simulations, only 2 incidents (simulation #6 and #10) occurred in the highway, while the other eight occurred in off/on ramps roads. In order to understand the cascading influence of the occurrence of a congestion and the potential of predicted congestions, we show a drill down analysis of the detected events in simulation #10 as a function of time and vicinity locations. A short description of the results is given for the other simulations.

5.2.2.2.1 Detailed analysis of simulation number 10

An incident happens at 18:12 and lasts for 21.5 minutes [18:12- 18:33] nearby location 4052. See specific part in the Rocade highway in Figure 40 below.

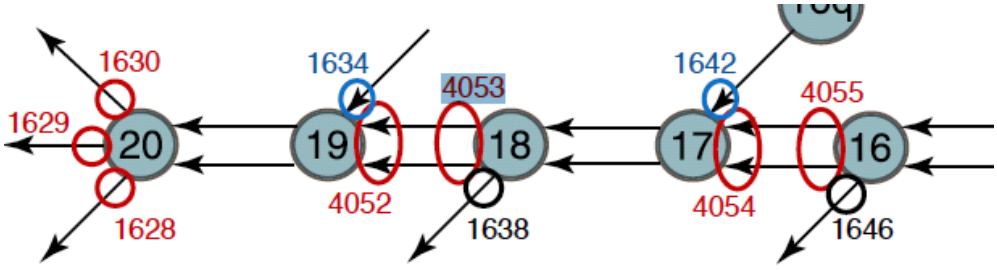

Figure 40: Zoom into simulation #10

The following congestions and trends situations result from the incident as detailed in Table 7 and are analyzed as follows. Note that the *PredictedTrend* derived event is shown as *Trend(certainty value)*.

- A first *Congestion* situation/derived event is emitted at 18:20:15 at location 4052. A second at 18:25:15 and a third at 18:30:15. Note that according to our design, we emit one derived event every 5 minutes (if a clear congestion doesn't take place before) once we have at least 15 readings over the threshold.
- Thereafter, the congestion at 4052 starts to decrease (still not cleared) but it causes a build-up and a *PredictedTrend* situation downstream in location 1629 at 18:37:45. Intuitively, the vehicles that started to leave 4052 produce a new build-up in 1629.
- From 18:39:00 until 18:41:45 we derive six *PredictedTrend* events with different certainty values in 4052 as a new build-up starts to form at this location. Note that the policy for the *PredictedTrend* event states that we emit a new *PredictedTrend* event each time the pattern is satisfied until either a *Congestion* or a *ClearCongestion* takes place. In our case a *Congestion* actually happens (as predicted) at 18:45:15 that closes the *PredictedTrend* temporal window.
- At 18:45:15 the buil-up at 4052 starts to influence the downstream location 1630 and a *PredictedTrend* event with relatively low certainty is detected at that location.
- The Congestion at 4052 at 18:45:15 causes a *PredictedCongestion* detection at 18:50:30 at location 4055 (upstream) and a *Congestion* at 4054 at 18:51:00 (also upstream).
- At 4055 three *PredictedTrend* derived events with increasing certainty values (0.326, 0.550, and 0.763) indicate a high probability of a *Congestion* which actually takes place at 18:56:30.
- The incident reported indeed ends after 21 minutes, however a new *Congestion* situation for the same location is detected at 18:45:15 which is not cleared up until the simulation ends (no *ClearCongestion* event is fired).

Table 7: Sequencing of derived events (situations) for incident #10

Sensor id / time	1629	1630	4052	4053	4054	4055
18:20:15			Congestion			
18:25:15			Congestion			
18:30:15			Congestion			
18:35:00			Trend(0.288)			
18:37:45	Trend(0.531)					
18:39:00			Trend(0.269)			
18:39:15			Trend(0.500)			
18:39:30			Trend(0.731)			
18:39:45			Trend(0.88)			
18:41:30			Trend(0.250)			
18:41:45			Trend(0.474)			
18:45:15		Trend(0.303)	Congestion			
18:50:30			Congestion			Trend(0.326)
18:50:45						Trend(0.550)
18:51:00					Congestion	Trend(0.763)
18:55:00			Congestion		Congestion	

18:56:00	Congestion
18:56:30	Congestion
18:59:15	Trend(0.302)

5.2.2.2.2 Brief analysis of simulations 2-6

Without the loss of generality, we show below snippets of our results that stress the findings described above for simulation #10. For each simulation we show the time, location, and derived event around the time the incident annotated takes place. For the *PredictedTrend* event we also give the corresponding certainty attribute. We color code the rows in a way that each color denotes a different seuqnce of *PredictedTrend* events followed by a *Congestion* event.

Simulation #1

18:50:00 4052 PredictedTrend (0.583)
18:53:30 4052 Congestion
18:54:45 4052 PredictedTrend(0.745)
18:57:45 4055 PredictedTrend(0.322)
18:58:45 4052 Congestion
18:59:15 4054 Congestion
18:59:45 4138 PredictedTrend (0.300)

We predict the impending *Congestion* in 3 cases, in two of which it deteriorates to an actual congestion state , and perhaps, could be avoided if taken action beforehand.

Simulation # 2

18:49:00 4052 PredictedTrend (0.503)
18:49:15 4052 PredictedTrend (0.64)
18:53:00 4053 PredictedTrend (0.319)
18:53:45 4052 Congestion
18:55:00 4052 PredictedTrend (0.244)
18:58:00 4055 PredictedTrend (0.365)
18:58:45 4052 Congestion
18:59:00 4053 Congestion
18:59:00 4054 Congestion

Again, *PredictedTrend* events are able to accurately forecast actual congestions a few minutes in advance (at 4052 and 4053).

Simulation #3

18:47:45 4052 PredictedTrend(0.326)
18:48:00 4052 PredictedTrend (0.549)

18:48:15	4052	PredictedTrend (0.758)
18:48:30	4052	PredictedTrend (0.895)
18:51:00	4053	PredictedTrend (0.334)
18:53:30	1630	PredictedTrend (0.304)
18:53:45	4052	Congestion
18:53:45	1630	PredictedTrend (0.553)
18:57:00	4055	PredictedTrend (0.291)
18:57:15	4055	PredictedTrend (0.510)
18:57:30	4055	PredictedTrend (0.767)
18:58:45	4054	Congestion
18:59:00	4052	Congestion
18:59:15	4053	Congestion
18:59:15	4166	PredictedTrend (0.230)
18:59:30	4166	PredictedTrend (0.478)

It can nicely be seen that in location 4052 there is four forecasted events that end up with a congestion at 18:59. Another sequence appears for location 4053 as a result of the propagation from 4052.

Simulation #4

18:53:45	4052	Congestion
18:53:45	4054	PredictedTrend (0.371)
18:57:00	4055	PredictedTrend (0.351)
18:57:15	4055	PredictedTrend (0.583)
18:58:30	4052	Congestion
18:59:00	4054	Congestion
18:59:00	4166	PredictedTrend (0.694)
18:59:15	4166	PredictedTrend (0.874)

As before, the congestion identified in 4054 impacts a upstream location in which a build-up starts (location 4055).

Simulation #5

18:44:30	1630	PredictedTrend(0.523)
18:44:45	1630	PredictedTrend (0.747)
18:48:30	4052	PredictedTrend (0.743)
18:48:45	4052	PredictedTrend (0.886)
18:53:50	4052	Congestion
18:57:39	4055	PredictedTrend (0.367)
18:58:30	4052	Congestion
18:58:30	4054	Congestion
18:58:30	4138	PredictedTrend (0.294)

18:58:45 4053 Congestion

Again, a congestion is identified after a forecasted event which impacts the flow in nearby sensors.

Simulation #6

18:09:30 4054 Congestion
18:13:30 4054 Congestion
18:18:30 4054 Congestion
18:25:00 4054 PredictedTrend (0.234)
18:27:00 1629 PredictedTrend (0.357)
18:30:30 1629 PredictedTrend (0.345)
18:39:30 1629 PredictedTrend (0.356)
18:49:00 4052 Congestion
18:50:15 4053 PredictedTrend (0.306)
18:50:30 4053 PredictedTrend (0.554)
18:53:00 4054 PredictedTrend (0.348)
18:53:30 4052 Congestion
18:54:15 4053 Congestion
18:54:45 3810 PredictedTrend (0.368)
18:55:00 3810 PredictedTrend (0.588)
18:58:45 4052 Congestion
18:58:45 4054 Congestion
18:59:00 4053 Congestion
18:59:00 1629 PredictedTrend (0.354)
18:59:15 1629 PredictedTrend (0.574)
18:59:15 4166 PredictedTrend (0.278)

In this simulation, an incident happens at 18:03 and lasts 17:20 minutes (until ~18:30).

As a result of the incident, an immediate congestion is built up at location 4054. Once the vehicles start moving after a while, a new congestion is identified upstream and we identify a new build-up at 1629. Later on there is the regular wave of congestion reflected in the *PredictedTrend* events at sensors 3810 and 4166.

5.2.2.2.3 Summary of results

Our findings are consistent among all the simulations. First, we were able to detect all congestions resulting from the simulated (annotated) incidents. Furthermore, we were able to detect more congestions as they happened in the simulations and indicated by the sudden drops of speed and high increase in density values. Moreover, *PredictedTrend* situations were detected and emitted which even caused *Congestion* situations a few minutes later. Note that we are running the CEP module in isolation so we don't have any feedback from the decision making module or any action taken that can help in alleviating the potential congestion.

Congestions or even predicted congestions have an impact upstream, and therefore forecasting a congestion in upstream locations when a congestion is detected in a downstream location can help in clearing up the whole area.

However, we still need to refine and validate the level of certainty that actually indicates a congestion, i.e. from which certainty value we should take an action (a congestion is very likely to happen). One time we get a congestion after certainty value of 0.763 and one time after 0.474 while we didn't get a congestion after certainty value of 0.88. There are some fluctuations in the numbers (see Table 7) which should be further analyzed and comprehended. To this end, we ran and analyzed a second set of simulations as detailed in the next section.

5.2.2.3 Recall and precision

In order to explore the quality of our results we ran a second set of simulations which comprised of:

- 20 simulations with annotations of congestions. All simulations last an hour.
- The annotations of congestions include the location and the time the congestion is detected.
- Other characteristics as file format and pre-processing of data remained the same, apart of the additional field for annotation of congestion (a Boolean field, having 1 for a congestions and 0 otherwise).

First, we checked the quality of our *Congestion* pattern against the annotated data. First, we checked the proportion of detections by our EPA that were annotated in the data as congestions (*precision*) and second, the proportion of congestions we were able to detect out of all the annotated congestions (*recall*). In all our simulations our precision was 100%, while the average recall over all the simulations was 72%. This can be easily explained: the rule implemented has been given to us by the domain expert, who is the one to identify the congestions in the simulations, thus giving a perfect precision. However, when implementing the pattern we applied a "stricter" criterion for the rule: we took into account not just the average *speed* critical thresholds , as was done in the simulations, but also *density thresholds*, therefore we have a less success rate in the recall of the results, i.e., there were annotations of congestion in the data that we "missed". When we "relaxed" the pattern and run the same rule as in the simulations we were able to detect all congestions with a perfect score in both precision and recall.

As a second step (as in our previous series of tests), we aimed at checking a more interesting question, that is, whether the inclusion of uncertainty aspects enables us to predict a congestion in the high way before it reaches critical thresholds, as opposed to detecting it once it happens. As before, we addressed this question by having two EPNs, once including uncertainty aspects and the other one without uncertainty, i.e. deterministic; and running the tests twice, one time for each EPN (with and without uncertainty). The deterministic case served as the "ground truth", as we knew at this stage that all our congestions have been detected correctly. The precision of our results indicates the proportion of congestions we were able to predict (in other words, *PredictedTrend* pointed out correctly to a congestion), whereas the recall indicates the proportion of congestions we were able to detect out of all the annotated congestions (in other words, *PredictedTrend* pointed out correctly out of all congestions). Important to note that we used a threshold of 0.6 in the *certainty* attribute to determine whether to

consider *PredictedTrend* as a congestion. In other words, only *PredictedTrend* alerts with a certainty value larger than 0.6 were considered in our calculations of precision and recall.

Table 8 summarizes our findings. As can be seen the average precision is 90.75% and the average recall is 75.05%. In addition we can see that sometimes a very high score in precision comes along with a lower score in recall (e.g., simulation #17, precision = 96.15 and recall = 51.85).

In general, the results indicate that the *PredictedTrend* pattern is a very good estimator of congestions to occur a few minutes ahead in time, thus enabling the system to take proactive actions in order to alleviate these congestions. However, lower scores in recall indicate that there are other situations that cause congestions which are not detected by our pattern. Further analysis on both real and generated data of these congestions that have not been "caught" by our pattern shows that these situations are characterized by "jumping data", meaning, the values of speed and density tend to jump thus not satisfying the increasing build-up which is required in our pattern. We are currently investigating these "jumping" cases to see if we can identify some common behavior/pattern.

Table 8: Summary of recall and precision results

Simulation	Precision	Recall
1	97.61	90.90
2	80	93.15
3	81.6	95.2
4	87.88	85.56
5	84.88	83.62
6	83.33	92.22
7	76.12	81.72
8	89.25	93.22
9	88.61	90.27
10	98	68.99
11	96.77	67.11
12	85.42	89.39
13	96	57.33
14	88.46	56.94
15	100	51.25
16	96.43	61.11
17	96.15	51.85
18	96.77	60.87
19	96.67	58.54
20	95	71.76
Average	90.75	75.05

6 Performance evaluation

The main metric of the system that we are interested in is the latency, as in the credit card fraud detection use case, this is the main constraint. We will follow the same approach for measuring latency as taken in the prototype performance evaluation analysis (refer to D6.5 Second Integrated Prototype[7]), that is: Latency is defined as the elapsed time between the detection time of the last input event required for a pattern matching and the corresponding detection time of the output event. For example: assuming that the (derived) event D is defined as a sequence of events (E1, E2, E3) then the latency is measured as the time period since an instance of E3 arrives into the system till the emission of the corresponding instance of D. Of course, this definition does not work for all event patterns. For instance, it is not applicable for 'absence' event patterns, or any other event patterns triggered by expiration of a time window. Therefore, in order for this definition to hold we will analyze the performance of PROTON on a set of "applicable" patterns with *IMMEDIATE* evaluation policy.

The conceptual view of the performance testing approach is presented in Figure 41. Input events are injected into PROTON run-time engine from an input events file. A module called *Analyzer* records all the events – both input and derived ones. For every event, the analyzer registers the detection times. The output of the analyzer is a test log which is processed later by the *Stats* utility. For every output event the *Stats* utility computes the processing latency (i.e.; the time between the last event in the matching set arriving to PROTON till the derivation).

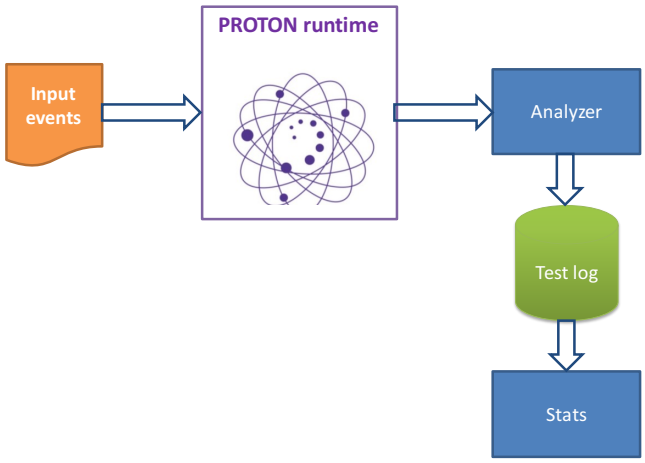

Figure 41: PROTON performance testing conceptual overview

In order to correlate the derived event with the latest input event that triggered the derivation, we leverage the new feature of PROTON that enables the derivation of the matching set events along with their attributes as part of the derived event payload (see Section 3). Thus, given an instance of the derived event, it is straightforward to compute the latency as defined above.

[7] Available at: http://speedd-project.eu/sites/default/files/D6.1_amended_and_D6.5_-_Architecture_Design_of_SPEEDD_Prototype_-_v2.1.pdf

6.1 Performance testing configuration

As aforementioned, the performance testing has been carried out in a set of "applicable" patterns in the credit card fraud detection use case, as it possesses the main latency constraint. The CP-EPN (see Section 4.1) is composed of 8 EPAs that perform the following operations: TREND (EPA1- EPA2); COUNT (EPA3-EPA5 + EPA8); and SEQ (EPA6-EPA7). Therefore, without the loss of generality, the application test is composed of three EPAs: EPA1 (*IncreasingAmounts*), EPA3 (*FlashAttack* with IMMEDIATE policy), and EPA4 (*MultipleMaxATMWithdrawals*) that are the same patterns found in the real data set. That is, we have a representative EPA of each type except the SEQ. The reasons are twofold: First, as it has been mentioned, these are the patterns that were found in the real data set. Second, the SEQ operator performance is heavily affected by the policies applied (as sometimes, we need all permutations of the events that can trigger a derivation). Therefore in order to decouple potential problems we decided to focus first on aggregators and trend operators.

The dataset consisted of 30K events injected at a non-uniform rate with peak of 100events/sec, randomly distributed over the context of credit card pan.

The performance testing was executed on a machine with the following configuration:

- CPU: Intel® Core ™ i7-4800MQ @ 2.70GHz -- Cores : Dual Core
- RAM: 16.0 GB
- OS: Windows 7 (64-bit)

6.2 Performance test results

Figure 42 shows PROTON's performance behavior, while the initial processing latency is low; it constantly increases over time until it reaches the values of **140,000 milliseconds** with an average latency of **11,186.4 milliseconds**.

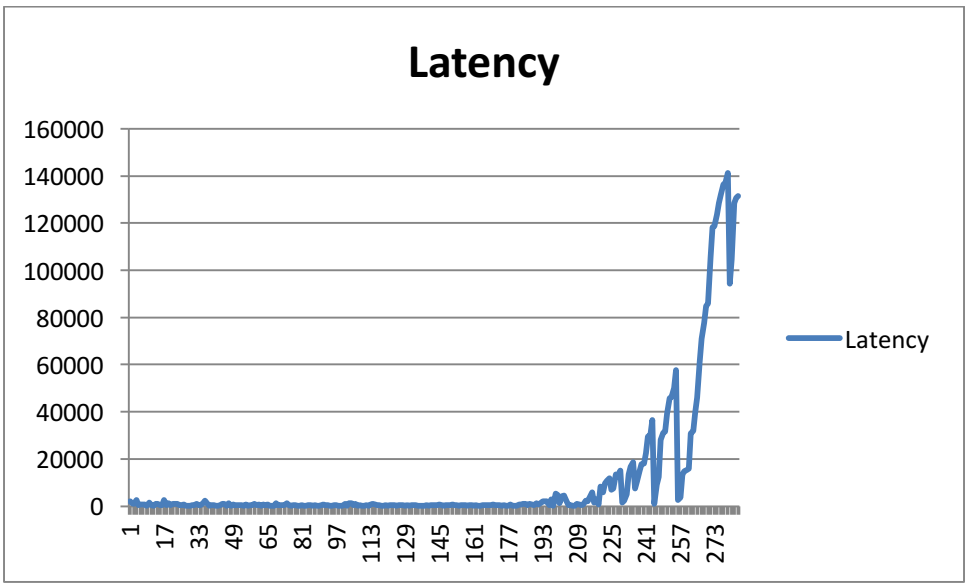

Figure 42: End-to-end latency in milliseconds

6.3 Analysis of results

The poor latency performance, in particular latency increasing over time, can be a result of gathering of large state in artifacts where state is relevant (such as EPAs) or possible delays/bottlenecks in execution. Obviously, we can also have these two in combination. We have investigated these two factors as explained below

6.3.1 EPAs latency

It is worthwhile to note that implementation in PROTON of almost all EPA operators is incremental: for aggregators only the intermediate calculations are kept (e.g., intermediate SUM, MAX, MIN values).Therefore, adding an additional event to the state is O(1) – adding its own value to the intermediate calculation. For TREND, the approach is the same – intermediate min/max trend value is stored in the EPA state, and therefore adding a new event only requires update of this single value. The exceptions to this rule are SEQ and ALL, since depending on the policies they might require iteration over all existing state of the EPA.

As our application implementation includes only aggregators and trend operators, it can be easily seen that the high latency does not stem from the event processing agents gathered state, as these only perform incremental calculations. As mentioned earlier, theoretically, due to their implementation, the only two operators in PROTON that can cause possible delays (depending on the applied policies), and can be (potentially) optimized in their internal implementation are SEQUENCE and ALL (logical AND). However these are not part of our EPN.

Therefore, we can conclude that there is no effect of the accumulated state in the EPAs on the high latency, leaving us with searching for bottlenecks at execution time. As will be explained below, this conclusion was later verified in profiling tools where it has been seen that performance bottlenecks are based in resource contention between threads executing context partitioning, and not in EPAs.

6.3.2 Bottlenecks during execution

In order to monitor and isolate the bottlenecks in the execution, we have used *VisualVM*[8] tool and the profiling options provided by the *JProfiler*[9] tool, specifically the memory profiling and thread monitoring options to monitor potential resource contentions and locks. Using the *locking graphs* option in the tool enabled us to reach the following conclusions regarding the main reasons for the poor latency performance:

Major bottleneck stems from synchronization calls in the evaluations of context segmentation expressions - As can be seen from Figure 43 (snapshot from VisualVM), major bottleneck stems from execution threads blocked on access to a monitor (object under contention). The higher the injection rate (the injection rate of input events increases over time reaching a pick of 100 events/second), more threads are created for handling the input events, and more blockage occurs. We can see that at highest injection rates, the threads are blocked on this monitor for 96% of the time.

Figure 43: Thread monitoring of the application during run

Farther analyses of the blocked execution thread stacks allowed us to see that the monitor the threads were waiting on, is the *EepExpression* object used to calculate segmentation context value. Specifically this happens when "evaluate" method of this object is invoked (see Figure 44).

[8] https://visualvm.java.net/
[9] https://www.ej-technologies.com/products/jprofiler/overview.html

```
com.ibm.hrl.proton.expression.eep.EepExpression.evaluate(com.ibm.hrl.proton.metadata.epa.basic.IDataObject) (line: 129)
com.ibm.hrl.proton.context.metadata.ComposedSegmentation.getSegmentationValue(com.ibm.hrl.proton.runtime.event.interfaces.IEventInstance) (line: 72)
com.ibm.hrl.proton.context.management.CompositeContextInstance.processContextParticipant(com.ibm.hrl.proton.runtime.event.interfaces.IEventInstance) (line:
com.ibm.hrl.proton.context.facade.ContextServiceFacade.processEventInstance(com.ibm.hrl.proton.runtime.timedObjects.ITimedObject, java.lang.String, java.lar
com.ibm.hrl.proton.eventHandler.EventHandler.handleEventInstance(com.ibm.hrl.proton.runtime.timedObjects.ITimedObject, java.lang.String, java.lang.String) (lin
com.ibm.hrl.proton.agentQueues.async.ConsumerWorkItem.run() (line: 46)
com.ibm.hrl.proton.server.workManager.WorkManagerFacade$1.run() (line: 43)
java.util.concurrent.ThreadPoolExecutor.runWorker(java.util.concurrent.ThreadPoolExecutor$Worker) (line: 1145)
java.util.concurrent.ThreadPoolExecutor$Worker.run() (line: 615)
java.lang.Thread.run() (line: 745)
```

Figure 44: The blocking monitor

In many applications, most of the EPAs use the same segmentation context throughout the application. For instance, in our case of credit card fraud detection, this segmentation context is based on the *credit card pan* (i.e., credit card number). All the three EPAs in this application share the same segmentation context of *credit_card_pan*. This context's metadata, along with the parsed expression for the evaluation of segmentation value are stored in a single context metadata object. The problem with this approach is that during the phase where the *context service* (see Figure 45) evaluates the value of the segmentation context per incoming event, it then locks the expression object for the duration of the evaluation (synchronized access in the *evaluate* method). This caused major blocking of all threads partitioning the input events into context subgroups. We have seen that with time, all the threads were blocked in this particular lock 96% of time.

Figure 45 depicts the order of the steps in PROTON run-time with the arrival of new events. The threads evaluating context segmentation value are locked on the expression object, resulting in long waits of input events for processing, and also causes starvation of the thread pool for the submitting threads.

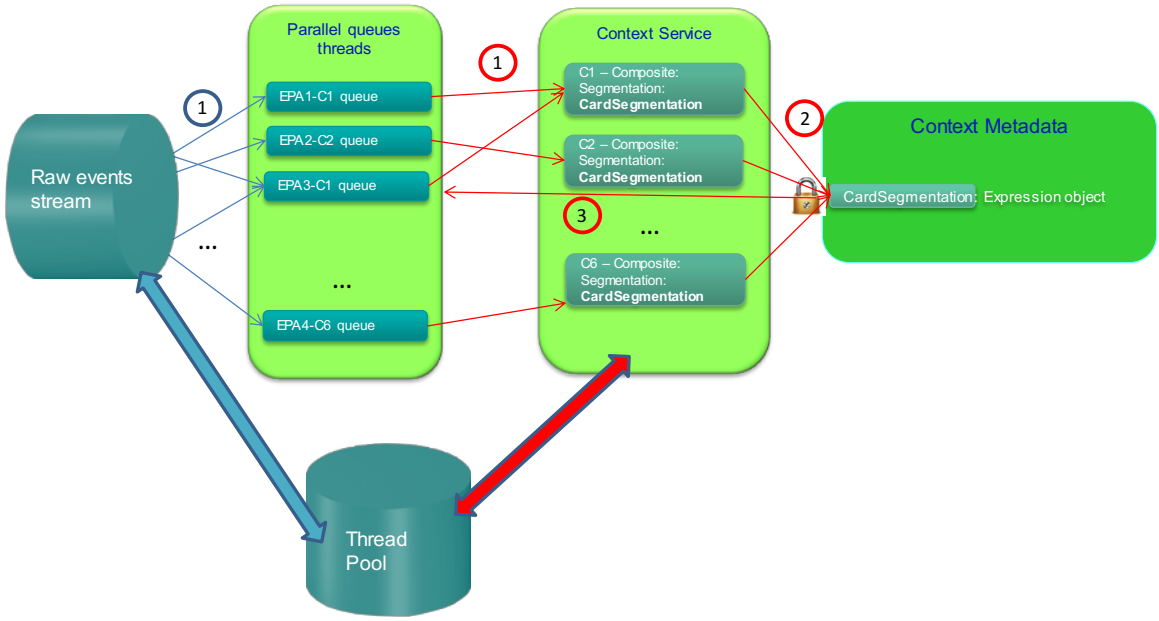

Figure 45: Evaluation of context segmentation expressions steps in PROTON's run-time

Description of the flow in Figure 45:

There are two types of tasks using threads from the same thread pool: the initial evaluation of input events according to the application's metadata and determination which agent/context this event is relevant for and submission for processing, and the evaluation of events by context service and partitioning to context instances. The order of input event processing is:

First flow: Input event is partitioned among agent-context queues for parallel processing. (1) This submission is parallel (gathering routing metadata in order to determine which agent and context is relevant) and is done by threads from a common thread pool.

Second flow: the events are consumed from the queues by the context service (1), (again using the threads available from the same thread pool), and partitioned according to their segementation value to context partitions (2+3). This partitioning process requires access to the segmentation context metadata (2) and evaluation which is performed in a synchronized block, since this action changes the object. Therefore, these steps (denoted in red lines in the figure) are blocking and take a lot of time on wait for this object.

As a consequence of the first waiting problem, a secondary problem of thread starvation has emerged. As we have seen in Figure 45, the input submitting threads used the same fixed-size thread pool as the processing threads, and this thread pool initial size was rather small (50 threads). As a result, the input submitting tasks waited on the pool until the blocking threads could finish their work and be returned to the pool. This caused an additional latency penalty.

After corrective actions have been carried out to solve these main two problems (see next Section 6.4), two additional minor issues were detected: logging overhead and overhead resulting from writing to a console as explained next.

Logging overhead - We used the *logger.debug("message")* statement to output debug information. This statement often requires *to string* conversion of contained message consisting of deep objects, such as Event (Event objects include all attributes and their values), or Context (Context objects include, for example, segmentation partitions values). As per logger implementation, the *logger.debug()* method outputs the message only when the logging level is appropriately configured to *DEBUG*. However, the message construction takes place in any case, even if the logging level is high. The message construction in case of such complex objects can be a costly operation.

Overhead resulting from writing to a console - Additional blocking occurred when some threads used *System.out PrintStream* to log messages to the console. The *println()* method is thread-safe in the used JVM implementation, and since it was found in heavily used portion of the code (context partition evaluation) it has significant effect on the latency of each thread.

6.4 Corrective actions

To overcome the issues found during the performance analysis, we have performed the following changes in PROTON's code base:

Instead of performing expression evaluation using the same expression object and synchronizing on the evaluation code, we now copy the expression object to local state of each thread (local variable in the point of expression evaluation) and perform the evaluation on this local object instead of a shared object, therefore eliminating the need of synchronization and increasing of thread pool size.

In addition, we have changed the fixed thread pool to a cached thread pool, more suitable for many short-lived tasks like in our case.

To deal with the unnecessary construction of String messages in the *logger.debug* statements, we wrapped the statement with assertion making sure it is only invoked if logging levels are indeed set to *DEBUG* as shown in the snippet below.

```
if (logger.isDebugEnabled())
{
    logger.debug("initializeCompositeContextInstances: getting context definitions from context metadata
}
```

In addition, we removed outputs to console, and used the *logger.debug* statements instead.

6.5 Performance improvements

A second performance evaluation analysis has been conducted after the changes aforementioned running the same application and dataset.

Latency has improved significantly (see Figure 46) – moving to an average of **139.8 milliseconds** (with an average matching set size of 3.2 events) and max peak latency of **2,000 milliseconds** at peak injection time (around 100 events per sec). This shows **80 times** improvement in the average latency and **70 times** improvement in peak latency comparing to our previous version Moreover, as can be also seen from the diagram, there is no more trend of increasing latency in the course of time.

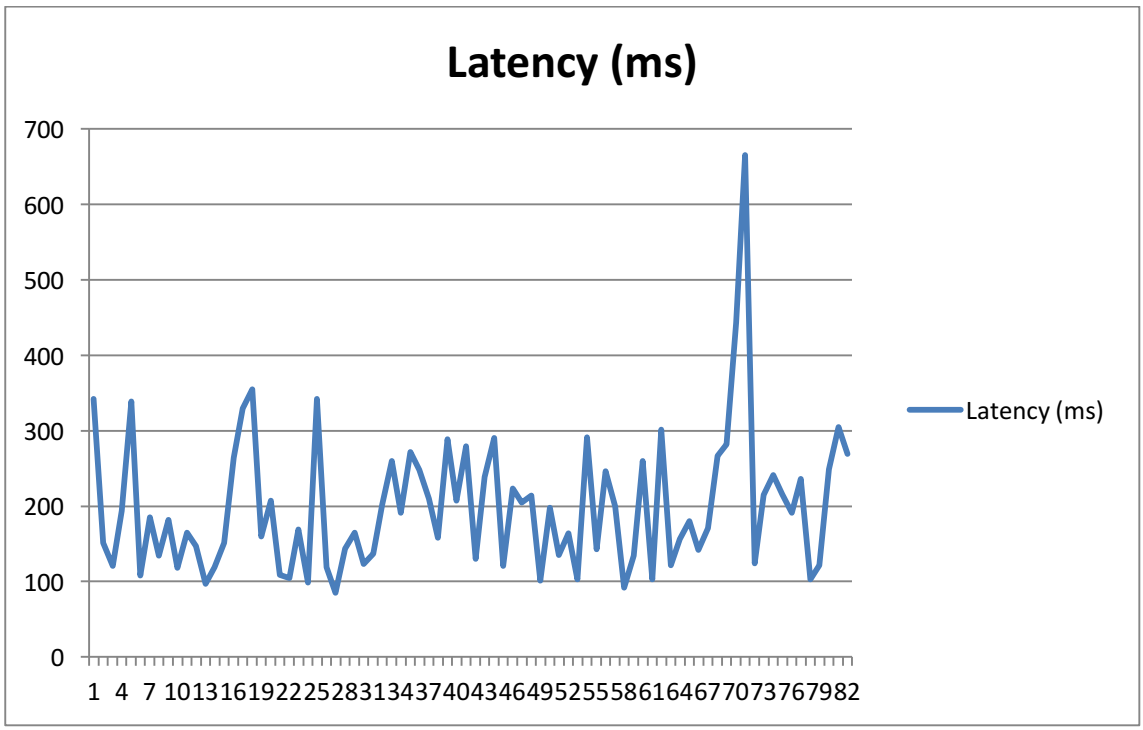

Figure 46: End-to-end latency in milliseconds after changes in PROTON's code base

6.6 Summary and future steps

Our performance analysis revealed a number of issues that caused poor latency results. Major cause was caused by synchronization calls in the evaluation of context segmentation expressions. Secondary causes are: thread starvation (initial thread pool size relatively small and using pool which is more suited for a finite number of long running tasks instead of extendable pool for many short-lived tasks), logging overhead, and the overhead due to writing to a console. Corrective actions have been introduced into the code base and propagated to all threads of PROTON (including the ProtonOnStorm open source version). Table 9 summarizes the improvements achieved in the latency measurements in PROTON as a result of SPEEDD developments.

Table 9: Latency performance before and after corrective actions

	Before (millisecs)	After SPEEDD developments (millisecs)
Maximum latency	140,000	2000
Average latency	11,186	140

Future steps include the investigation of ProtonOnStorm version when replacing the code base of PROTON by this updated run-time version.

7 Semantic translation from event calculus to PROTON's programming model

Events rules or patterns of an event driven application can be given or defined by an expert, learnt by a machine learning component, or both. In SPEEDD we strive to compose a set of event rules that are given by domain experts in the domain (traffic management and credit card fraud detection) and also learnt by the machine learning component in the project. At this stage, PROTON only processes patterns given by domain experts, but we have started to look into augmenting the existing set of rules by the work done by machine learning. As a first step, we analyzed some of the event calculus examples from the fraud detection use case and manually translated into an EPN which can be converted into a JSON definition file for PROTON in a straightforward manner. Some insights and rules during this process have been identified. During the third year of the project we aim at continuing this work and investigating more examples in order to compose a methodology on how to perform the semantic translation between the two languages. The output of this work will be a unified set of rules incorporated in the JSON definition file of PROTON.

7.1 Event Calculus examples

We have analyzed the principles of the event calculus semantics with the NCSR team in the project and identified some patterns which we applied to three different event calculus rules given to us by NCSR. The three examples are described in the next sub-sections.

7.1.1 Previous Transactions

```
1.2 initiatedAt(prev_trx(Pan, Country)=exists, T) WHEN
        happensAt(trx(_TrId, _CNP, _AmountEU, Pan, _ExpDate, Country), T)
```

Description: If transaction `trx` happens for card in country at time `T`, then set that, from now on, there exists a previous transaction (for this specific card (Pan) in this specific `Country`)

7.1.2 Amount

```
happensAt(amount_level(Pan, Country, low), T) WHEN
        happensAt(trx(_TrId, _CNP, AmountEU, Pan, _ExpDate, Country), T) AND
        AmountEU =< 20.0
happensAt(amount_level(Pan, Country, medium), T) WHEN
        happensAt(trx(_TrId, _CNP, AmountEU, Pan, _ExpDate, Country), T) AND
        AmountEU > 20.0 AND
        AmountEU =< 100.0
happensAt(amount_level(Pan, Country, high), T) WHEN
        happensAt(trx(_TrId, _CNP, AmountEU, Pan, _ExpDate, Country), T) AND
        AmountEU > 100.0
```

Description: Simple filtering on amount:

- If amount spent (AmountEU) below 20, then set amount_level as low
- If between 20 and 100, then set as medium
- If above 100, then set as high

7.1.3 First TRX in country high

```
happensAt(first_trx_high(Pan, Country), T) WHEN
        happensAt(amount_level(Pan, Country, high), T) AND
        NOT holdsAt(prev_trx(Pan, Country)=exists, T)
```

Description: If there is no previous transaction for this card (NOT holdsAt(prev_trx(Pan, Country)=exists, T) and the amount_level of this transaction is high (see above rule, where AmountEU > 100), then we recognise that this first transaction has a high amount (first_trx_high).

7.2 Conclusions

At this stage we can deduce the following semantic rules in order to translate from event calculus to PROTON's language.

- **Fluents** are derived events, and therefore we can transform each fluent to an EPA
- **Events** can be transformed to Filters EPAs (as there is no context associated, just conditions on the events)
- We don't have negation per-se. One way to workaround this is by using COUNT + policies as in our example.
- AND between events can be translated into an ALL operator
- initiatedAt and terminatedAt (or any other condition for termination) – temporal context in PROTON.
- Note: we disregard the confidence values for the time being.
- Note: EC doesn't address pattern policies per se, and we decide upon them on a case by case basis (see EPAs below)

Following these rules we can convert the three event calculus rules into the following EPAs

7.2.1 EPA1: amount_level

Event recognition process:

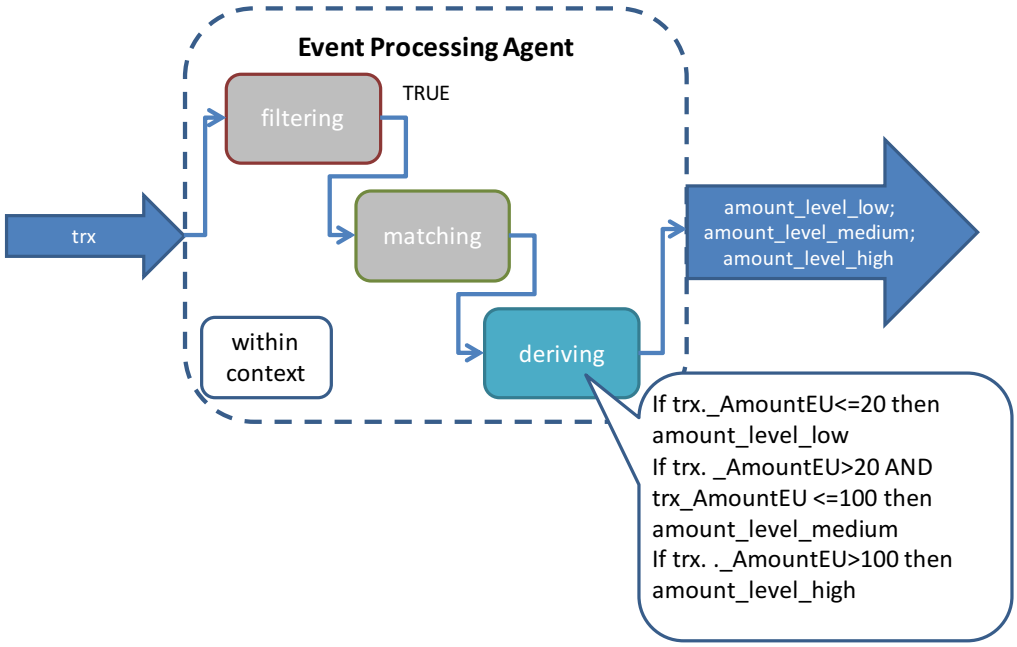

Figure 47: Event recognition process for amount_level EPA

Pattern policies: N/A

Context: N/A

Note that we actually don't need a specific EPA in this case and can use the "high" case in the filter of the pre_trx EPA (see below), but for the sake of clarity and in the case we would like to use also the "medium" and "low" cases.

7.2.2 EPA2: prev_trx

Event recognition process:

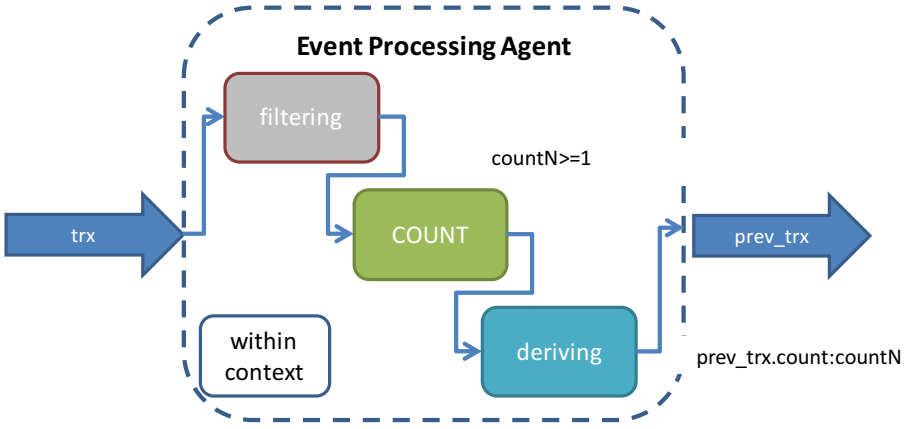

Figure 48: Event recognition process for prev_trx EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	UNRESTRICTED	FIRST	REUSE

Context:

Segmentation: by Pan & Country

Initiator policy: ALWAYS (from Startup)

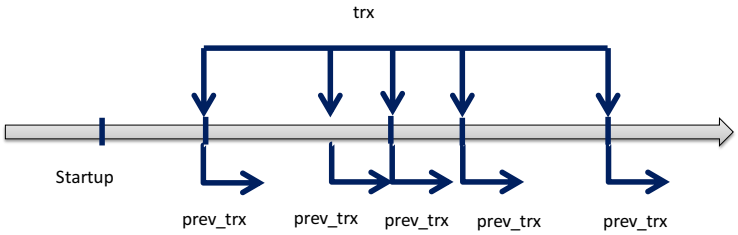

Figure 49: Event recognition process for prev_trx EPA

7.2.3 EPA3: first_trx_high

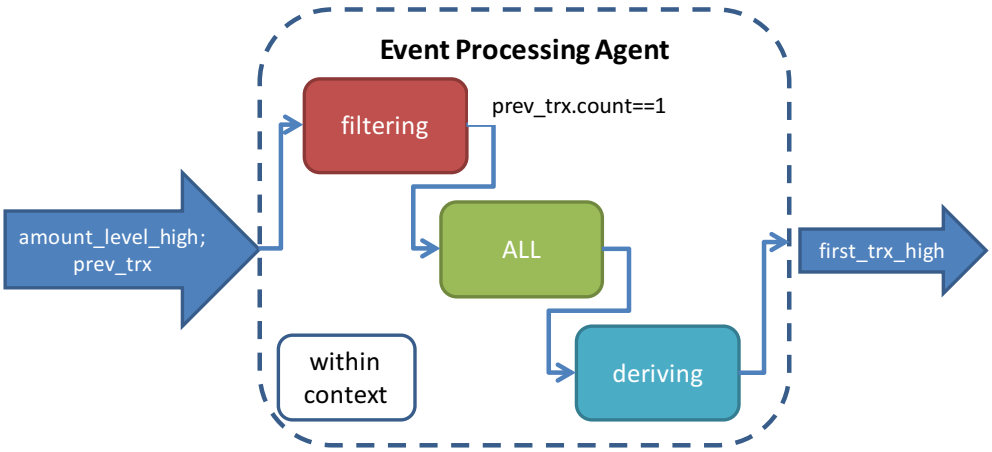

Figure 50: Event recognition process for first_trx_high EPA

Pattern policies:

Evaluation	Cardinality	Repeated	Consumption
IMMEDIATE	SINGLE	FIRST	REUSE

Context:

Segmentation: by Pan & Country

Initiator policy: IGNORE

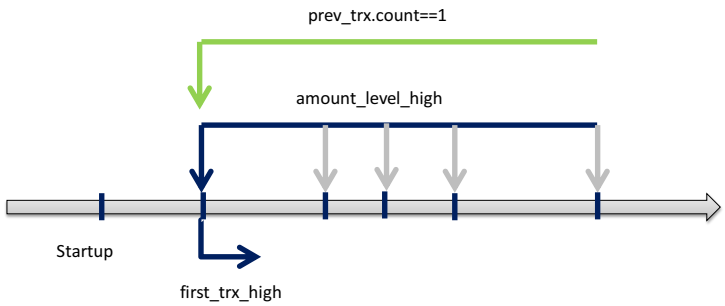

Figure 51: Event recognition process for first_trx_high EPA

7.2.4 Event processing network

The resulting EPN for these three EPAs is depicted in Figure 52.

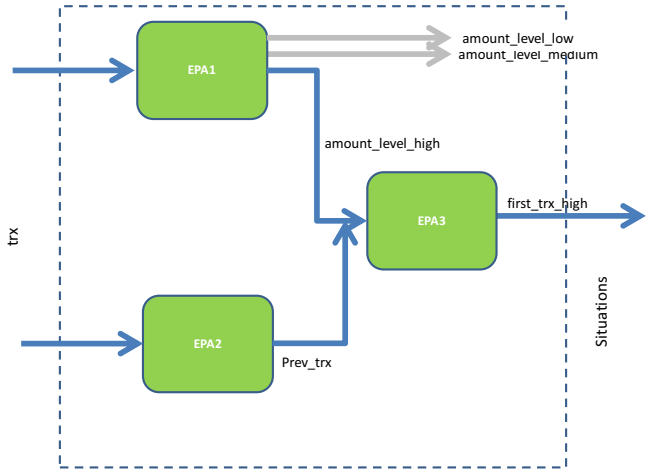

Figure 52: first_trx_high EPN

8 Summary and future steps

In this document, we present a second version of the complex event module under uncertainty in SPEEDD. The report mainly focuses on the two new applications and the evaluation results for the two use cases of the project: the credit card fraud detection and the traffic management. In addition we describe developments made to the PROTON tool during the second year of the project to address the project requirements, and progress made in the interaction between the machine learning part and the run-time.

Regarding the evaluation of the two applications, as the main objective is to assess the potential impact of the inclusion of uncertainty aspects into event-driven applications; we run our experiments using an "uncertain" EPN and its "certain" counterpart (the baseline). Our results show that both use cases gained benefits in the uncertain case:

- In the fraud detection use case – we are able to derive a potential fraud before the certain case and therefore, have the chance to block the credit card (alternatively, deny the next transaction) **before** further transactions take place. Let's recall that if a transaction is marked as accepted, nothing can be done afterwards if the transaction happens to be fraudulent. Therefore, the main benefit of the uncertain application is financial, as we prevent from further transactions to take place.
- In the traffic use case - we are able to forecast a congestion before it actually happens and therefore **proactive actions** can be taken to alleviate the congestion. We showed that a *PredictedCongestion* event is emitted 3 to 4 minutes before a *Congestion* is detected and therefore a remediation action is possible.

However, performance analysis showed poor latency. In this second version of the document, a new section on performance analysis and description of refactoring of the code to improve performance in the stand-alone version of PROTON is included. As a result, latency measurements have been marginally improved. Important to note that these improvements hold only in the stand alone version, and better measurements are expected to be achieved in the ProtonOnStorm version.

Having a new version of the run-time engine, further recall and precision have been studied for the traffic use case showing that we can predict with a high level of success a future congestion.

However, there are some limitations in our applications and we plan to focus on overcoming these limitations during year three of the project.

In the credit card use case:

- Having no access to the sample data. The main drawback is the fact that the tests are made at Feedzai's premises. The entire process required a great deal of "logistics" from both sides that made the process more complex and delayed the actual run of the application. Needless to say,

each tool requires a certain level of expertise when working with it, and PROTON is not an exception to this rule.

- There are inherent limitations in our application that primarily stem from the fact that we don't rely on enrichment of the data by customer profiling. This is one of the main steps done in Feedzai's system. As long as we cannot get this information, will be very difficult to marginally improve our results. Still, we plan to improve our EPN with more fined tuned patterns and thresholds.
- As pointed out, our results are inconclusive with regards to their correctness. Further analysis on the data with the assistance of Feedzai needs to be carried out.

In the traffic use case:

- Our results indicate that these are significantly influenced by the thresholds applied. We used "rules of thumb" or thresholds by visually looking at the data. We intend to learn the thresholds and values for the *Sigmoid* function from the historical data to better reflect real situations in our next EPN version. Moreover, the level of certainty that actually indicates a congestion should be validated, i.e. from which certainty value we should take an action (a congestion is very likely to happen).
- One of the main advantages of CEP engines, is the capability of easily include heterogeneous events. We started our work in this direction with the incorporation of the weather API. We plan to continue this direction in next year.

9 References

[1]. O. Etzion and P. Niblet. *Event processing in action*. Manning, 2010

[2]. Y. Kim and F. Hall. *Relationships Between Occupancy and Density Reflecting Average Vehicle Lengths*. Transportation Research Record: Journal of the Transportation Research Board, 1883:85–93, January 2004.

[3]. M. Cassidy and B. Coifman. *Relation Among Average Speed, Flow, and Density and Analogous Relation Between Density and Occupancy*. Transp. Res. Rec. J. Transp. Res. Board, vol. 1591, pp. 1–6, Jan. 1997.

[4]. C. Canudas de Wit, F. Morbidi, L. L. Ojeda, A. Y. Kibangou, I. Bellicot, and P. Bellemai. *Grenoble Traffic Lab: An Experimental Platform for Advanced Traffic Monitoring and Forecasting [Applications of Control].* IEEE Control Syst., vol. 35, no. 3, pp. 23–39, Jun. 2015.

[5]. G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli. Introducing uncertainty in complex event processing: model, implementation, and validation. Computing, pages 1–42, 2014. ISSN 0010-485X. doi: 10.1007/s00607-014-0404-y.

Scalable Data Analytics Scalable Algorithms, Software Frameworks and Visualisation ICT-2013.4.2a

Project **FP7-619435 / SPEEDD**

Deliverable **D3.2**

Distribution **Public**

http://speedd-project.eu/

Second Version of Event Recognition and Forecasting Technology – Part II

Evangelos Michelioudakis, Anastasios Skarlatidis, Elias Alevizos, Alexander Artikis, Georgios Paliouras, Nikos Katzouris, Christos Vlassopoulos, Ioannis Vetsikas

Status: Revised (Version 2.0)

August 2016

Project

Project ref.no.	FP7-619435
Project acronym	SPEEDD
Project full title	Scalable ProactivE Event-Driven Decision making
Porject site	http://speedd-project.eu/
Project start	February 2014
Project duration	3 years
EC Project Officer	Stefano Bertolo

Deliverable

Deliverabe type	report
Distribution level	Public
Deliverable Number	D3.2
Deliverable title	Second Version of Event Recognition and Forecasting Technology – Part II
Contractual date of delivery	M22 (November 2015)
Actual date of delivery	August 2016
Relevant Task(s)	WP3/Task 3.1
Partner Responsible	NCSR “D”
Other contributors	
Number of pages	74
Author(s)	Evangelos Michelioudakis, Anastasios Skarlatidis, Elias Alevizos, Alexander Artikis, Georgios Paliouras, Nikos Katzouris, Christos Vlassopoulos, Ioannis Vetsikas
Internal Reviewers	
Status & version	Revised
Keywords	Uncertainty, event calculus, statistical relational learning

Contents

List of Tables

Executive Summary

This document presents the advancements made in event recognition and forecasting technology of the SPEEDD project, in order to reason about events and learn event definitions over large amounts of data, as well as under situations of uncertainty.

SPEEDD implements event recognition methods (also known as event pattern matching or event pattern detection systems), in order to extract useful information, in the form of events, by processing time-evolving data that comes from various sources (e.g., various types of sensor, network activity logs, ATMs, transactions, etc.). The extracted information — recognized and/or forecasted events — can be exploited by other systems or human experts, in order to monitor an environment and respond to the occurrence of significant events. Event recognition methods employ rich representation that can naturally and compactly represent events with complex relational structure, e.g., events that are related to other input/derived events with spatio-temporal constraints. Unfortunately, uncertainty is an unavoidable aspect of real-world event recognition applications and it appears to be a consequence of several factors. Consider for example, noisy or incomplete observations from road sensors, as well as imperfect definitions fraudulent activity. Under situations of uncertainty, the performance of an event recognition system may be seriously compromised. Another important characteristic of the SPEEDD project, is that machine learning algorithms must deal with large amounts of data that continuously evolves. As a result, the current base of event definitions may need to be refined or new events may appear. Therefore, the traditional approach of non-incremental batch machine learning algorithms cannot be applied in SPEEDD.

We begin by reviewing event recognition techniques that handle, to some extent, uncertainty. We examine automata-based and logic-based techniques, which are the most common ones, and approaches based on Petri Nets and Context-Free Grammars, which are less frequently used. A number of limitations are identified with respect to the employed languages, their probabilistic models and their performance, as compared to the purely deterministic cases. Based on those limitations, we highlight promising directions for future work.

We subsequently present a method for scalable incremental learning. In order to address the requirements imposed by the presence of uncertainty, we combine probabilistic and logic-based modeling for representing and reasoning about events and their effects under uncertainty. Specifically, we take advantage of the succinct, structured and declarative representation of the Event Calculus formalism, in order to formally express events and their effects. To handle the uncertainty, we employ the state-of-the-art probabilistic and relational framework of Markov Logic Networks. The combination of probabilistic and logical modeling has also the advantage of expressing event definitions with well defined probabilistic

and logical schematics and thus, we can employ state-of-the-art probabilistic reasoning and machine learning techniques. Specifically, we present our developed probabilistic structure learning method that exploits the background knowledge axiomatization to effectively constrain the space of possible structures by learning clauses subject to specific characteristics defined by these axioms. We employ an online strategy in order to effectively handle large training sets and incrementally refine the previously learned structure.

1

Introduction

1.1 History of the Document

Version	Date	Author	Change Description
0.1	3/11/2015	Anastasios Skarlatidis (NCSR)	Set up of the document
0.2	4/11/2015	All Deliverable Authors (NCSR)	Structure of the document
0.3	10/11/2015	All Deliverable Authors (NCSR)	Content adjusted: Event Recognition Survey
0.4	10/12/2015	All Deliverable Authors (NCSR)	Content adjusted: Structure Learning for MLN
0.5	15/1/2016	All Deliverable Authors (NCSR)	Content adjusted: Experimental Evaluation
1.0	23/2/2016	All Deliverable Authors (NCSR)	Content adjusted: Coclusions
2.0	25/7/2016	All Deliverable Authors (NCSR)	Final version after internal review

1.2 Purpose and Scope of the Document

This document presents the progress of the SPEEDD project with respect to event recognition and forecasting under uncertainty, as well as the current advancements to probabilistic inference and machine learning for event definitions. Furthermore, the presented work identifies the research directions that will be pursued in the second year of the project.

The reader is expected to be familiar with Complex Event Processing, Artificial Intelligence and Machine Learning techniques, as well as the general intent and concept of the SPEEDD project. The target relationship is:

- SPEEDD researchers
- SPEEDD audit

SPEEDD emphasises to scalable event recognition, forecasting and machine learning of event definitions for Big Data, under situations where uncertainty holds. This document presents the current

advancements and discusses the scientific and technological issues that are being investigated in Work-Package 3.

1.3 Relationship with Other Documents

This document is related to project deliverable D3.1 “Event Recognition and Forecasting Technology” that presents previous work on event recognition and machine learning for the SPEEDD prototype. Furthermore, deliverables D6.1 and D6.5 “The Architecture Design of the SPEEDD Prototype and Second Integrated Prototype ” which present the SPEEDD prototype architecture. Finally, updated versions of deliverables D7.1 and D8.1 that outline the requirements and the characteristics of the “Proactive Credit Card Fraud Management” and “Proactive Traffic Management” project use cases, respectively.

2

Event Recognition under Uncertainty: A Survey

2.1 Introduction

Systems for Complex Event Recognition (CER) accept as input a stream of time-stamped, simple, derived events (SDE)s. A SDE ("low-level event") is the result of applying a computational derivation process to some other event, such as an event coming from a sensor. Using SDEs as input, CER systems identify complex events (CE)s of interest— collections of events that satisfy some pattern. The "definition" of a CE ("high-level event") imposes temporal and, possibly, atemporal constraints on its subevents, i.e. SDEs or other CEs. Consider, for example, the recognition of attacks on computer network nodes given the TCP/IP messages. A CER system attempting to detect a Denial of Service attack has to identify (as one possible scenario) both a forged IP address that fails to respond and that the rate of requests is unusually high. Similarly, in the maritime monitoring domain, in order to detect an instance of illegal fishing, a CER system has to perform both some basic geospatial tasks, such as estimating whether a vessel is moving inside a protected area, and temporal ones, like determining whether a vessel spent a significant amount of time in this area.

The SDEs arriving at a CER system almost always carry a certain degree of uncertainty and/or ambiguity. Information sources might be heterogeneous, with data of different schemas, they might fail to respond or even send corrupt data. Even if we assume perfectly accurate sensors, the domain under study might be difficult or impossible to model precisely, thereby leading to another type of uncertainty. Until recently, most CER systems did not make any effort to handle uncertainty (it is instructive to see the relevant discussion about uncertainty in Cugola and Margara (2011)). This need is gradually being acknowledged and it seems that this might constitute a major line of research and development for CER.

The purpose of this survey is to present an overview of existing approaches for CER under uncertainty. Since this field is relatively new, without a substantial number of contributions coming from researchers directly involved with CER, we have chosen to adopt a broader perspective and include methods targeting activity recognition and scene understanding on image sequences coming from video sources. Although activity recognition is a field with its own requirements, it is related closely enough to CER so that some of the ideas and methods applied there might provide inspiration to CER researchers as well. However, it is not our intention to present a survey of video recognition methods and we have selectively chosen those among them that we believe are closer to CER (for a survey of activity recognition methods from video sources, see Vishwakarma and Agrawal (2013)). We have used two basic

criteria for our choice (applied to the CER methods as well). First, we require that the method employs some kind of relational formalism to describe activities, since purely propositional approaches are not sufficient for CER. Second, we require that uncertainty be handled within a probabilistic framework, since this is a framework that provides clear and formal semantics. In this respect, our work is related to previously conducted comparisons within the field of statistical relational learning, both theoretical De Raedt and Kersting (2003); Jaeger (2008); Muggleton and Chen (2008) and practical Bruynooghe et al. (2009). Note also that the related field of query processing over uncertain data in probabilistic databases/streams is covered in other surveys (e.g., in Wang et al. (2013a)) and, therefore, we will not include such papers in our survey.

Running example

Throughout the remainder of the survey, we are going to use a running example in order to assess the presented approaches against a common domain. Our example comes from the domain of video recognition. We assume that a CER engine receives as input a set of time-stamped events, derived from cameras recording a basketball game. However, we need to stress that our input events are not composed of raw images/frames and that the task of the CER engine is not to perform image processing. We assume availability of algorithms that can perform the corresponding tasks, such as object recognition and tracking. Therefore, our SDEs consist of events referring to objects and persons, like *walking*, *running* and *dribbling*. The purpose is to define patterns for the recognition of some high-level long-term activities, e.g., that a *matchup* between two players or a *double-teaming* is taking place.

Structure of survey

The structure of the survey is as follows: Section 2.2 discusses briefly the types of uncertainty that may be encountered in a CER application. In Section 2.3 we present certain criteria based on which we included (or excluded) papers from our survey and in Section 2.4 we discuss the dimensions along which a proposed solution for handling uncertainty may be evaluated. Section 2.5 presents the reviewed approaches. Finally, Section 2.6 summarizes them in a tabular form and comments on their limitations. Some open issues and lines of potential future work are also identified.

2.2 Uncertainty in Event Recognition

Understanding uncertainty in its different types and various sources is crucial for any CER system that aspires to provide an efficient way of handling it. The ideal CER system would be capable of handling all types of uncertainty within a unified and mathematically rigorous framework. However, this is not always possible and the current CER systems are still far from achieving such an ideal. Different domains might be susceptible to different types of uncertainty, while different CER engines employ various methods for responding to it, ranging from the ones that simply ignore it to those that use highly complex, fully-fledged, dynamic, probabilistic networks. In this section, we give a brief description and classification of the various types of uncertainty that may be encountered by a CER system. For further discussion, see Wasserkrug et al. (2006); Artikis et al. (2012).

2.2.1 Data Uncertainty

The event streams that provide the input data to a CER engine can exhibit various types of uncertainty. In this section, we present the main types of uncertainty that may be found in incoming event streams.

Incomplete Data Streams

One type of uncertainty is that of incomplete or missing evidence. A sensor may fail to report certain events, for example due to some hardware malfunction. Even if the hardware infrastructure works as expected, certain characteristics of the monitored environment could prevent events from being recorded, e.g. an occluded object in video monitoring or a voice being drowned by stronger acoustic signals.

Corrupt Data

The events of the input stream may have a noise component added to them. In this case, events may be accompanied by a probability value. There are many factors which can contribute to the corruption of the input stream, such as the limited accuracy of sensors or distortion along a communication channel. Another distinction which might be important in certain contexts is that between stochastic and systematic noise, e.g. the video frames from a camera may exhibit a systematic noise component, due to different light conditions throughout the day.

When noise corrupts the input event stream, a CER system might find itself in a position where it receives events asserting contradictory statements. For example, in a computer vision application which needs to track objects, such as that of our running example, if there are multiple software classifiers, one of them may assert the presence of an object (e.g., the ball) whereas another may indicate that no such object has been detected.

Finally, when a CER system needs to learn the structure and the parameters of a probabilistic model from training data, quite often the data are inconsistently annotated. Therefore, the rules to be learned have to incorporate this uncertainty and carry a confidence value.

2.2.2 Pattern Uncertainty

Besides the uncertainty present in the input data, a noise-tolerant CER system should also be able to handle cases where the event patterns are not precise or complete.

Due to lack of knowledge or due to the inherent complexity of a domain, it is sometimes impossible to capture exactly all the conditions that a pattern should satisfy. It might also be preferable and less costly to provide a more general definition of a pattern which is easy to implement rather than trying to exactly determine all of its conditions. A pattern with a wider scope, which does not have to check multiple conditions, may also be more efficient to compute and, in some cases, this performance gain could be more critical than accuracy. In both cases, we cannot infer an event with certainty and a mechanism is required to quantify our confidence.

For example, a rule for determining when a team is attempting an offensive move might be defined as a pattern in which one of the team's players has the ball and all other players are located in the opponent team's half-court. However, the same pattern could also be satisfied when a player is attempting a free throw or for an out-of-bounds play. Depending on our requirements, we might or might not want to include all of these instances as cases of offensive moves. Defining all of these sub-cases would require more refined conditions, something which is not always possible. Yet, we might be still interested in capturing this pattern and provide a confidence value.

2.3 Scope of the survey

Before presenting our framework and evaluation dimensions, we explain the rationale behind our choices and clarify some basic conceptual issues, which we deem important from the point of view of CER.

2.3.1 Probabilistic models

A seemingly simple method to handle uncertainty is to ignore or remove noise through pre-processing or filtering of the data, thus facilitating the use of a deterministic model. Other methods are available as well, such as possibilistic reasoning, conflict resolution (accept data according to the trustworthiness of a source) and fuzzy sets. For example, in Shet et al. (2007) and Shet et al. (2011) a logic-based method is proposed, which employs logic programming and handles uncertainty using the Bilattice framework Ginsberg (1988). Another example is the work presented in Ma et al. (2010), where the Dempster-Shafer theory is used in order to take into account the trustworthiness of heterogeneous event sources. We focus on probabilistic models because they provide a unified and rigorous framework and the bulk of research on CER under uncertainty employs such models.

2.3.2 Time representation

Some approaches, especially those employing dynamic graphical models, resort to an implicit representation of time, whereby time slices depend on (some of) the previous slice(s), without taking into account time itself as a variable. Useful as this solution might be in domains characterized by sequential patterns, such as activity recognition in video, there are other cases in CER where time constraints need to be explicitly stated. Although we include in our survey some approaches with an implicit time representation, our focus will mostly be on probabilistic relational methods with explicit time representation.

2.3.3 Relational models

A substantial proportion of the existing probabilistic models are propositional by nature, as is the case with many probabilistic graphical models, such as simple Bayesian networks. Probabilistic graphical models have been successfully applied to a variety of CER tasks where a significant amount of uncertainty exists. Especially within the machine vision community, they seem to be one of the most frequently used approaches. Since CER requires the processing of streams of time-stamped SDEs, numerous CER methods are based on sequential variants of probabilistic graphical models, such as Hidden Markov Models (HMM) Rabiner and Juang (1986) and their extensions (e.g., coupled Brand et al. (1997), Dynamically Multi-Linked Gong and Xiang (2003) and logical Hidden Markov Models Kersting et al. (2006)), Dynamic Bayesian Networks Murphy (2002) and Conditional Random Fields Lafferty et al. (2001).

As far as Hidden Markov Models are concerned, since they are generative models, they require an elaborate process of extracting the correct independence assumptions and they perform inference on the complete set of possible worlds. Moreover, their first-order nature imposes independence assumptions with regard to the temporal sequence of events (with only the current and the immediately previous states taken into account) that might not be realistic for all domains. On the other hand, Conditional Random Fields are discriminative models, a feature which allows them to avoid the explicit specification of all dependencies and, as a consequence, avoid imposing non-realistic independence assumptions Vail et al. (2007); Wu et al. (2007); Liao et al. (2005). However, both Hidden Markov Models and Conditional Random Fields assume a static domain of objects (with the exception of logical Hidden Markov Models Kersting et al. (2006)), whereas a CER engine cannot make the same assumption, since it is not possible to determine beforehand all the possible objects that might appear in an input stream from a dynamic and evolving environment. Additionally, the lack of a formal representation language makes the definition of structured CEs complicated and the use of background knowledge very hard. From a CER perspective, these issues constitute a severe limitation, since rules for detecting CEs often require relational and

hierarchical structures, with complex temporal and atemporal relationships. For these reasons, we do not discuss Hidden Markov Models and Conditional Random Fields in a more detailed manner. Instead, we focus our investigation on methods with relational models.

2.4 Evaluation Dimensions

In this section, we provide a general framework for the discussion of the different approaches and establish a number of evaluation dimensions against which the strengths and weaknesses of each method may be assessed. We follow the customary division between representation and inference. In other words, we are interested in what kind of knowledge a system can encode (representation) and what kind of queries it can answer (inference). Although learning in general is a very active research area, we have decided not to include a detailed discussion about the learning capabilities of the examined approaches in our survey. The reason is that almost none of the systems touches upon this subject (Sadilek and Kautz (2012) is one exception). Instead, we will try to draw some conclusions about the performance of each system, taking into account the difficulties in making performance comparisons, as explained in the relevant section below.

2.4.1 Representation

A simple unifying event algebra

We begin our discussion of representation by introducing a basic notation for CER. For a more detailed discussion of the theory behind CER, we refer readers to Luckham (2001), Etzion and Niblett (2010a), Cugola and Margara (2010). Following the terminology of Luckham (2001), we define an event as an object in the form of a tuple of data components, signifying an activity and holding certain relationships to other events by time, causality and aggregation. An event with N attributes can be represented as $EventType(ID, Attr_1, \ldots, Attr_N, Time)$, where $Time$ might be a point, in case of an instantaneous event, or an interval during which the event happens, if it is durative. Notice, however, that, when timepoints are used, some unintended semantics might be introduced, as discussed in Paschke (2006). For our running example, events could be of the form $EventType(EventID, PlayerName, UnixTime)$ and one such an event could be the following: $Running(987865, Antetokounmpo, 19873294673)$. In CER, we are interested in detecting patterns of events among the streams of SDEs. Therefore, we need a language for expressing such pattern detection rules. For example, by using ';' as the sequence operator, the pattern

$$EventType_1(ID_1, A_1^1, \ldots, A_N^1, T_1);$$
$$EventType_2(ID_2, A_1^2, \ldots, A_N^2, T_2)$$

would serve to detect instances where an event of type $Type_1$ is followed by an event of type $Type_2$. An example would be:

$$Running(987865, Antetokounmpo, 19873294673);$$
$$Jumping(987653, Antetokounmpo, 19873294677);$$
$$Dunking(987234, Antetokounmpo, 19873294680)$$

In the remainder of the survey, for convenience, we omit the ID from events.

Based on the capabilities of existing CER systems and probabilistic CER methods, we adopt here a simple event algebra. Formalisms for reasoning about events and time have appeared in the past, such as the Event Calculus Kowalski and Sergot (1986); Cervesato and Montanari (2000) and Allen's

Interval Algebra Allen (1983b, 1984), and have already been used for defining event algebras (e.g. in Paschke and Bichler (2008)). With the help of the theory of descriptive complexity, recent work has also identified those constructs of an event algebra which strike a balance between expressive power and complexity Zhang et al. (2014). Our event algebra will be defined in a fashion similar to the above mentioned efforts, borrowing mostly from Zhang et al. (2014); Cervesato and Montanari (2000).

The following list enumerates those operations that should be supported by a CER engine:

- $Sequence$: Two events following each other in time.
- $Disjunction$: Either of two events occurring, regardless of temporal relations. Conjunction (both events occurring) may be expressed by combining $Sequence$ and $Disjunction$.
- $Iteration$: An event occurring N times in sequence, where $N \geq 0$.
- $Negation$: Absence of event occurrence.
- $Selection$: Select those events whose attributes satisfy a set of predicates/relations, temporal or otherwise.
- $Production$: Return an event whose attribute values are a possibly transformed subset of the attribute values of its sub-events.
- $Windowing$: Evaluate the conditions of an event pattern within a specified time window.

Below, we present the syntax of the event algebra:

$$
\begin{array}{lll}
ce ::= & sde & | \\
& ce_1 \ ; \ ce_2 & | \ Sequence \\
& ce_1 \ \vee \ ce_2 & | \ Disjunction \\
& ce^* & | \ Iteration \\
& \neg \, ce & | \ Negation \\
& \sigma_\theta(ce) & | \ Selection \\
& \pi_m(ce) & | \ Production \\
& [ce]_{T_1}^{T_2} & \ Windowing \text{ (from } T_1 \text{ to } T_2)
\end{array}
$$

where $\sigma_\theta(ce(v_1, \ldots, v_n))$ *selects* those ce whose variables v_i satisfy the set of predicates θ and $\pi_m(ce(a_1, \ldots, a_n))$ *returns* a ce whose attribute values are a possibly transformed subset of the attribute values of a_i of the initial ce, according to a set of mapping expressions m. $Conjunction$ may be written as $ce ::= ce_1 \wedge ce_2 ::= (ce_1; ce_2) \vee (ce_2; ce_1)$.

For example, a traveling violation, occurring when a player who has possession of the ball takes more than two steps without dribbling, could be defined as follows:

$$
\begin{aligned}
&traveling(P^{'}, T^{'}) ::= \pi_{P'=P,T'=T_3}(\sigma_{T_3-T_1<5 \ seconds}(\\
&(hasBall(P, T_1) \wedge takesStep(P, T_1) \wedge \neg dribbling(P, T_1)) \ ; \\
&(hasBall(P, T_2) \wedge takesStep(P, T_2) \wedge \neg dribbling(P, T_2)) \ ; \\
&(hasBall(P, T_3) \wedge takesStep(P, T_3) \wedge \neg dribbling(P, T_3)) \ ))
\end{aligned}
$$

where we include the five-second rule and assume that $hasBall$, $takesStep$ and $dribbling$ are all SDEs.

The above algebra is simple, but expressive, defining temporal constraints between events. E.g., in the above rule about $traveling$, the $sequence$ operator (;) implies that $T_2 > T_1$ and $T_3 > T_2$. Note that some CER systems (e.g., the Chronicle Recognition System Dousson et al. (1993); Dousson (2002); Dousson and Le Maigat (2007)) allow the predicates θ to be applied directly to the attribute of time, as in the previous five-second rule. Throughout the remainder of the survey, we also adopt the selection policy of $skip-till-any-match$ (irrelevant events are ignored and relevant events can satisfy multiple "instances" of a rule) and the $zero-consumption$ policy (an event can trigger multiple rules).

The above syntax allows for the construction of event hierarchies, a crucial capability for every CER system. Being able to define events at various levels and reuse those intermediate inferred events in order to infer other, higher-level events is not trivial. Theoretically, every event language could achieve this simply by embedding the patterns of lower-level events into those at higher levels, wherever they are needed. However, this solution would result in long and contrived patterns and would probably incur heavy performance costs, since intermediate events would need to be computed multiple times. Moreover, there are multiple ways a system could handle the propagation of probabilities from low-level to high-level events and these differences can affect both the performance and the accuracy of the system.

Probabilistic Data

The event algebra defined above is deterministic. We now extend it in order to take uncertainty into account. As we have already discussed, we can have uncertainty both in the data and the patterns. As far as data uncertainty is concerned, we might be uncertain about both the occurrence of an event and about the values of its attributes. E.g., the ProbLog2 system Fierens et al. (2013) employs annotated disjunctions. Therefore, for a probabilistic event, we could write $Prob :: EventType(Value_1, \ldots, Value_N, Time)$, which means that this event with these values for its attributes might have occurred with probability $Prob$ and not have occurred at all with probability $1 - Prob$. In order to assigne probabilities to attribute values, e.g. for two different values of $Attribute_1$, we could write

$$\begin{aligned} Prob_1 :: \quad & EventType(Value_1^1, \ldots, Value_N, Time) \ ; \\ Prob_2 :: \quad & EventType(Value_1^2, \ldots, Value_N, Time) \end{aligned}$$

In this case, the sum of the two probabilities should not exceed the value of 1 and, if it is below 1, this means that there is also a probability of no occurrence at all, whose value is $1 - \sum_s Prob_s$. We assume that probabilistic events are represented as discrete random variables.

With respect to the probability space, a common assumption is that it is defined over the possible histories of the probabilistic SDEs. If SDEs are defined as discrete random variables, then one SDE history corresponds to making a choice about each of the SDEs among mutually exclusive alternative choices. The probability distribution is then defined over those SDE histories. E.g., if we have the following probabilistic SDEs

$$\begin{aligned} 0.8 :: Running(Antetokounmpo, 19873294673) \\ 0.6 :: Jumping(Antetokounmpo, 19873294677) \\ 0.7 :: Dunking(Antetokounmpo, 19873294680) \end{aligned}$$

then the probability space is composed of the 8 possible histories obtained through all the combinations of event (non-)occurrence. Choosing the history in which all events do occur would yield a probability of $0.8 * 0.6 * 0.7$, assuming that all SDEs are independent (which is not always the case).

Probabilistic Model

In addition to handling uncertain data, we also require probabilistic rules. Syntactically, we express a probabilistic rule by appending its probability value as a prefix, e.g.

$$Prob :: ce(A,T) ::= \pi_{A=A_2,T=T_2}(ce_1(A_1,T_1);ce_2(A_2,T_2))$$

where, if ce_2 occurred after ce_1, then ce occurred at T_2 with probability $Prob$. The probability space is extended to include the inferred CE in the event histories. A probabilistic rule should then be understood as defining the conditional probability of the CE occurring, given that its sub-events occurred and satisfied its pattern. The attribute values of this CE are those returned by the $production$ operator. If we need to define different probability values to the CE with different attribute values, we could use again the syntax of annotated disjunctions, e.g.

$$\begin{aligned} &Prob :: ce(T,A) ::= \\ &\pi_{T=T_2,A=A_2}(ce_1(T_1,A_1);ce_2(T_2,A_2)) \quad ; \\ &Prob^{'} :: ce(T,A) ::= \\ &\pi_{T=T_2,A=2*A_2}(ce_1(T_1,A_1);ce_2(T_2,A_2)) \end{aligned}$$

where the occurrence probability of ce, with its attribute A having value A_2, is $Prob$, whereas it may also have occurred with $A = 2 * A_2$ and probability $Prob^{'}$.

There are other ways to define the probability space and its semantics. For example, in the probabilistic programming literature it is common to use the possible worlds semantics for the probability space (e.g., in ProbLog Fierens et al. (2013)). The probability distribution is defined over the (possibly multi-valued) Herbrand interpretations of the theory, as encoded by the CE patterns. In this setting, we could assign non-zero probabilities even in cases where the rule is violated and we could end up with every Herbrand interpretation being a model/possible world. The existence of "hard" rules which must be satisfied excludes certain interpretations from being considered as models. When using grammars (and sometimes logic), the space might be defined over the possible proofs that lead to the recognition/acceptance of a CE.

2.4.2 Inference

In probabilistic CER, the task is often to compute the marginal probabilities of the CEs, given the evidence SDEs. Consider the following example:

$$P(offense(MilwaukeeBucks,[00:00,00:24])|SDEs)$$

where we want to calculate the probability that the team of $MilwaukeeBucks$ was on the offensive for the first 24 seconds of the game and we assume that $offense(team,[start,end])$ is a durative CE, defined over intervals and in terms of SDEs, such as $running$, $dribbling$, etc. Moreover, there are cases when we might be interested in performing maximum a posteriori (MAP) inference, in which the task is to compute the most probable states of some CEs, given the evidence SDE stream. A simple example from our running example is the query which asks about the most probable time interval during which an offense by a team is taking place:

$$I_{offense} = \underset{I}{argmax} P(offense(MilwaukeeBucks,I)|SDEs)$$

Another important dimension concerns the ability of a system to perform approximate inference. For all but the simplest cases, exact inference stumbles upon serious performance issues, unless several

simplifying assumptions are made. For this reason, approximate inference is considered essential. Certain systems also provide answers with confidence intervals and/or the option of setting a confidence threshold above which an answer may be accepted.

2.4.3 Performance

CER systems are usually evaluated for their performance in terms of throughput, measured as number of events processed per second and latency, as measured by the average time required to process an event. Less often, the memory footprint is reported. Standard benchmarks have not yet been established, although some work towards this direction has begun Mendes et al. (2013, 2009); Grabs and Lu (2012). Reporting throughput figures is not enough by itself, since there are multiple factors which can affect performance. E.g., the number of different event types in the SDE stream or the rule selectivity, i.e. the percentage of SDEs selected by a rule, are such factors (see Mendes et al. (2009) for a comprehensive list of performance affecting factors). When uncertainty is introduced, the complexity of the problem increases and other factors that affect performance enter the picture, such as the option of approximate inference.

Systems need to be evaluated along another dimension as well, that of accuracy. Precision and recall are the usual measures of accuracy. In some applications, recall is more important than precision and in others, the focus is on precision. F-measure is the harmonic mean of precision and recall. The issue of accuracy is of critical importance and is not orthogonal to that of performance. A system may choose to sacrifice accuracy in favor of performance by adopting techniques for approximate inference. Another option would be to make certain simplifying assumptions with respect to the dependency relationships between events so that the probability space remains tractable.

2.5 Approaches

Surprisingly, there have not been many research efforts devoted exclusively to the problem of handling uncertainty within the community of distributed event-based systems. The majority of research papers that could be deemed as relevant to our problem actually come from the computer vision community. Perhaps it is not much of a surprise if one takes into account the historical roots of CER systems. Stemming from the need to build more active databases and to operate upon streams of data that have a pre-defined schema, the problem of uncertainty, although present, was not as critical as in the case of efficiently processing events from real-world sensors. Moreover, for many of the domains where CER solutions were initially applied, the goal was to produce some aggregation results (averages, medians, etc.) from the input streams, which, in a sense, already constitute statistical operations.

Our analysis has identified the following classes of methods: automata-based methods, logic-based methods, probabilistic Petri nets and approaches based on context-free grammars.

2.5.1 Automata-based methods

Most research efforts targeting the problem of uncertainty in CER are based on extensions of crisp engines. Since many of these engines employ automata, it is not surprising that automata are one of the dominant approaches. In this section, we present these approaches. Compared to other methods, those based on automata seem better-suited to CER, since input events in CER are usually in the form of streams/sequences of events, similarly to strings of characters recognised by (Non-)Deterministic Finite Automata.

SASE Wu et al. (2006) and its extension, SASE+ Agrawal et al. (2008), which includes support for $Kleene$ closure, is an automata-based CER engine which has frequently been amended in order to support uncertainty Shen et al. (2008); Kawashima et al. (2010); Zhang et al. (2010); Wang et al. (2013b). The focus of SASE is on recognizing sequences of events through automata. For each CE pattern, a automaton is created whose states correspond to event types in the sequence part of the pattern (excluding possible negations). As the stream of SDEs is consumed, the automaton transitions to a next state when an event is detected that satisfies the sequence constraint. The recognized sub-sequences are pruned afterwards, according to the other non-sequence constraints (e.g., attribute equivalences), but, for some of these constraints, pruning can be performed early, while the automaton is active. Those SDEs that triggered state transitions are recorded in a data structure that has the form of a directed acyclic graph, called Active Instance Stack, allowing for quick retrieval of those sub-sequences that satisfy the defined pattern. SASE+ deviates somewhat from this scheme in that it employs NFA^b automata, i.e., non-deterministic finite automata with a buffer for storing matches. For the $skip-till-any-match$ selection policy, where all possible SDE combinations that match the pattern are to be detected, the automaton is cloned when a SDE allows for a non-deterministic action. For example, a SDE whose type satisfies a $Kleene$ operator, may be selected or ignored, in which case a new automaton is created.

Assume that our SDEs consist of events indicating whether a certain player holds the ball, is running, dribbling, shooting or jumping. Additionally, assume that we also have SDEs about whether the ball is in the net. Now consider a pattern detecting an assist from a player X to a player Y. This pattern could have the following simplistic definition:

$$\begin{aligned} assist(X,Y,T_3) ::= & hasBall(X,T_1); \\ & hasBall(Y,T_2); \\ & shooting(Y,T_3); \\ & ballInNet(T_4) \end{aligned} \tag{2.1}$$

where player X first has the ball, then player Y, who subsequently attempts a shot and finally it is detected that the ball is in the net. Note that, for convenience, we omit explicitly writing some production and selection predicates, like $X \neq Y$. We assume that the same variable symbols (like X in the CE and in the SDEs) refer to the same variable and different symbols to different variables (like X and Y). We also note that this is a simplistic definition, since the rule does not exclude cases where there might be intervening $hasBall$ SDEs between the two detected $hasBall$ SDEs. A more refined rule would have to make sure that the two detected $hasBall$ SDEs are consecutive, but we use this simplistic definition for demonstration purposes, in order to show the basic functionality of automata.

Suppose we have a stream of probabilistic SDEs, as the one shown in Table 2.1. The first and third columns correspond to timestamps in seconds, while the second and fourth columns show SDEs and CEs respectively. Each SDE has a probability prefix and its arguments correspond to players (here simply denoted as pN) and timestamps, except for the $ballInNet$ SDE, whose only argument is the timestamp. The $assist$ CEs have three arguments, two for the players involved and one for the timestamp. Ignoring for the moment the probabilities, the crisp version of SASE, for the $assist$ pattern (2.1), would construct an automaton and Active Instance Stacks, as shown in Figure 2.1. Since rule (2.1) is a simple sequence pattern, without iteration, the NFA simply proceeds to a next state when an event of the appropriate type arrives. The other alternative is to ignore an incoming event (self-loops) and wait for another event in the future to satisfy the pattern. By following the arrows in the Active Instance Stack, the events satisfying the pattern can be efficiently retrieved. SASE would recognize the sequence of checkmarked SDEs in Table 2.1, producing the $assist(p2,p3,6)$ CE. Note that rule (2.1), as it stands, would also recognize two other CEs, namely those including the $0.8 :: hasBall(p2,3)$ and $0.9 :: hasBall(p1,1)$. Here we focus only on the CE produced by the checkmarked SDEs, as shown by the thick arrows in Figure 2.1.

Time	Input	Time	Output
1	$0.9 :: hasBall(p1, 1)$		
1	$0.8 :: dribbling(p1, 1)$		
1	$0.95 :: running(p2, 1)$		
3	$0.8 :: hasBall(p2, 3)$		
4	$0.7 :: hasBall(p2, 4)$ ✓		
4	$0.7 :: dribbling(p2, 4)$		
5	$0.9 :: hasBall(p3, 5)$ ✓		
6	$0.85 :: shooting(p3, 6)$ ✓		
6	$0.95 :: jumping(p6, 6)$		
7	$0.9 :: ballInNet(7)$ ✓	7	$P_1 :: assist(p2, p3, 6)$
		7	$P_2 :: assist(p2, p3, 6)$
		7	$P_3 :: assist(p1, p3, 6)$
...			

Table 2.1: A stream of probabilistic SDEs from the basketball example

For the probabilistic versions of SASE, the issue is how to correctly and efficiently calculate the probability of the produced CEs. In all of these versions Shen et al. (2008); Kawashima et al. (2010); Zhang et al. (2010); Wang et al. (2013b), this probability is calculated by conceptualizing a probabilistic stream as event histories, produced by making a choice among the alternatives of each SDE. In our example, there are only two alternatives for each of the 10 SDEs in the example data stream – occurrence and non-occurrence –, hence 1024 event histories. In Kawashima et al. (2010); Wang et al. (2013b), each SDE is treated as having only these two alternatives. However, in other works, as in Shen et al. (2008), a SDE may have more alternatives, corresponding to different values for some of the arguments of the SDE. In Zhang et al. (2010), uncertainty about SDEs concerns their timestamps, which are described by a distribution, an issue not addressed in other works.

The probability of a CE could be calculated by enumerating all the histories, selecting those which satisfy the CE pattern and summing their probabilities. The probability of a history depends on the independence assumptions that each approach makes with respect to SDEs. Moreover, since a full enumeration is highly inefficient, optimization techniques are employed in order to calculate CE probabilities.

In the simplest case, all SDEs are assumed to be independent. In Kawashima et al. (2010), where this assumption is followed, a matching tree is gradually constructed with SDEs that trigger state transitions. By traversing the tree, the sequence of SDEs producing a CE and its probability can be retrieved in a straightforward manner, through multiplications and summation, since all SDEs are independent. For our example, the probability of the $assist(p2, p3, 6)$ CE would be $0.7 * 0.9 * 0.85 * 0.9$, i.e. the product of the SDE probabilities. In this approach, as more SDEs arrive, probabilities can only become smaller and, by defining a confidence threshold, certain branches of the tree may be early pruned.

In Shen et al. (2008), SDEs are again assumed to be independent, but a full enumeration is avoided by using a modified version of the Active Instance Stack, used in crisp SASE. A similar approach is used in Wang et al. (2013b), where the assumption of complete SDE independence is relaxed and some SDEs may follow a first-order Markov process. In this case, the edges of the Active Instance Stack are anno-

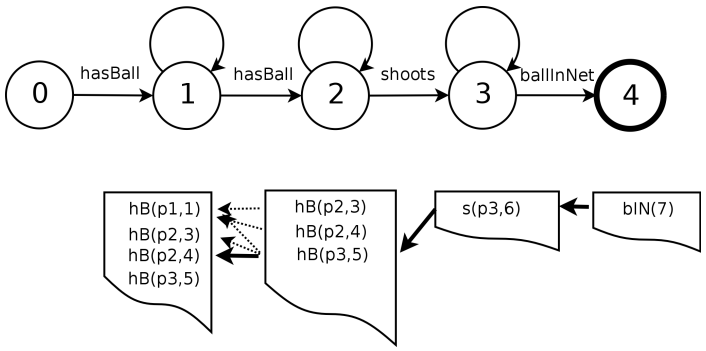

Figure 2.1: NFA and Active Instance Stack as constructed by SASE for rule (2.1) and example stream of Table 2.1. hB=$hasBall$, s=$shooting$, bIN=$ballInNet$.

tated with the conditional probabilities. If we assume that $P(ballInNet(7)|shooting(p3,6)) = 0.95$, then $P(ballInNet(7), shooting(p3,6)) = P(shooting(p3,6)) * P(ballInNet(7)|shooting(p3,6)) = 0.85 * 0.95$, hence $P(assist(p2,p3,6)) = 0.7 * 0.9 * 0.85 * 0.95$. Note that the conditional probability tables in this approach are based on event types, e.g., $P(ballInNet|shooting)$ for all $ballInNet$ and $shooting$). For every specific SDE dependent on another SDE, probabilities must be explicitly provided. In this work, hierarchies of CEs are also allowed.

In Zhang et al. (2010), the issue of imprecise timestamps is addressed, while all the other attributes have crisp values. Again, SDEs are assumed to be independent. Complete enumeration of all possible worlds is avoided by employing an incremental, three-pass algorithm through the events in order to construct event matches and their intervals. This work was extended in Zhang et al. (2014), which added $negation$ and $Kleene\ plus$ and allowed for user-defined predicates.

All of these SASE-based methods perform marginal inference and use confidence thresholds for pruning results that fall below them. Only one attempts to increase performance through distribution. To the best of our knowledge, the work in Wang et al. (2013b) is one of the very few developing a CER system which is both probabilistic and distributed (PADUA, described later, is the only other such method).

Non SASE-based approaches have also appeared. A recognition method that models activities using a stochastic automaton language is presented in Albanese et al. (2007). In this case, it is not the SDEs that are probabilistic, but the state transitions of the automaton, in a way similar to Markov chains. A possible world is now essentially defined over activity occurrences that are targeted for recognition, i.e. CEs. This method could not initially represent any temporal constraints. It was extended in Albanese et al. (2011), in order to identify situations that cannot be satisfactorily explained by any of the known CEs. In particular, the stochastic automaton model is extended with temporal constraints, where subsequent SDEs can occur within a user-defined temporal interval. Using possible-worlds based modeling, the method finds (partially) unexplained activities. This is the only automata-based method that can perform both marginal and MAP inference.

Another extension of Albanese et al. (2007) is the PADUA system, presented in Molinaro et al. (2014). PADUA employs Probabilistic Penalty Graphs and extends the stochastic automaton presented in Albanese et al. (2007) with noise degradation. The edges that connect subsequent events in a Penalty Graph, forming the structure of a CE, are associated with a probability (noise) value that degrades the belief of the CE when other events intervene. As a result, under such situations, the CE is being recognized, but with reduced probability. Besides event recognition itself, the method can find patterns of events that do not belong to the set of known CEs. Additionally, for purposes of scalability, Penalty

Graphs can be combined by merging common sub-Graphs, indexing and executing them in parallel.

The Lahar system Ré et al. (2008) constitutes one of the earliest proposals and is based on the Cayuga Demers et al. (2006) CER engine. Events are modelled by first-order Markov processes. The possible queries are categorized in three different classes of increasing complexity. For the first two types of queries, automata are used for recognition. For the most complex queries, in which variables are not shared among all of the conditions (e.g., as in rule (2.1), with X and Y), a version of the Probabilistic Relational Algebra Fuhr and Rölleke (1997) is used. A method which attempts to overcome the strict markovian hypothesis of Lahar and apply certain optimizations, such as early pruning, may be found in Chuanfei et al. (2010).

Automata-based methods focus on recognizing sequences of events, in which some of those events may be related, via their attributes, to other events of the sequence. In general, time representation is implicit. As a result, and with the exception of Albanese et al. (2011), they do not include explicit temporal constraints, such as concurrency or inequalities between timestamps, e.g., a constraint like $T_4 - T_1 \leq 24\ seconds$ in rule (2.1) to make sure than the recognized activity takes place within a single offense. $Windowing$ is the only temporal constraint allowed. Moreover, they only address the issue of data uncertainty (Albanese et al. (2011) is again the exception), lacking a treatment of other types of uncertainty, such as pattern uncertainty, and model relatively simple probabilistic dependencies between events.

To illustrate a case where concurrency and more complex dependencies may be required, consider a pattern trying to detect an attempted block by a defender from the stream of Table 2.1:

$$\begin{aligned} attempt\_block(Y,T) ::= & \sigma_{opponents(X,Y)}(\\ & shooting(X,T) \wedge \\ & jumping(Y,T) \wedge \\ & close(X,Y,T)\ ) \end{aligned} \tag{2.2}$$

where player X is shooting at the same time that player Y is jumping, the distance between them is small at that time (we assume image recognition can provide such information as $close$ SDEs) and the two players belong to different teams, i.e., they are $opponents$. Such a pattern would require explicit temporal constraints or, at least, an implicit constraint about concurrent events, a feature generally missing in automata-based methods. Moreover, $jumping$ is clearly dependent on $shooting$ (a player usually jumps at the same time or after another player shoots but not while all other players run), yet this dependence cannot be captured by assuming a Markov process that generated those events. Note also that the above rule makes use of the $opponents$ predicate, assuming that the engine can take into account such background knowledge that is not part of the SDE stream. Such knowledge is relatively easy to model in logic-based systems, but the automata-based ones presented above have no such mechanism.

Now assume we want to express a rule stating that if two players are $close$ to each other at the current timepoint, then they are likely to be close at the next timepoint (a first-order Markov assumption). This is not an event we would like to detect in itself, but domain knowledge which we would like our system to take into account. Such rules may be helpful in situations where SDEs may suddenly be missing, for example due to some sensor failure, but the activity has not ceased. We could introduce the following two rules:

$$1 :: close\_m(X,Y,T) ::= close(X,Y,T) \tag{2.3}$$

$$\begin{aligned} 0.6 :: close\_m(X,Y,T) ::= & \sigma_{next(T,T_{previous})}(\\ & close\_m(X,Y,T_{previous})\ ) \end{aligned} \tag{2.4}$$

and use the $close\_m$ predicate instead of $close$ in the definition of rule (2.2) for $attempt\_block$. The first of these rules simply transfers the "detection" probability of $close$ to that of $close\_m$ (rule probability is 1), whereas the second one expresses the Markov assumption. In automata-based methods where Markov assumptions are allowed, the conditional probabilities need to be provided for every "ground" pair of SDEs. Uncertain rules allow us to describe such dependencies in a more succinct manner, as "templates".

Assume also that we need some rules to detect maneuvers in which the offender attempts to avoid the defender. Two of these rules could be the following:

$$0.9 :: avoid(X, Y, T_2) ::= waiting(X, Y, T_1); \\ crossover\_dribble(Y, T_2) \tag{2.5}$$

$$0.7 :: avoid(X, Y, T_2) ::= waiting(X, Y, T_1); \\ running(Y, T_2) \tag{2.6}$$

where $crossover\_dribble$ and $waiting$ are assumed to be CEs detected by their respective rules.

In this case, we have a hierarchy of CEs, defined by probabilistic rules, starting with the SDEs, on top of them the $waiting$ and $crossover\_dribble$ CEs and finally the $avoid$ CE. An efficient mechanism for propagating probabilities among the levels of the CE hierarchy would be required. Among the presented methods, CE hierarchies are allowed only in Wang et al. (2013b). Moreover, combining rules would also be required, both for rules (2.3) – (2.4) and rules (2.5) – (2.6), i.e. functions for computing the probabilities of CEs with multiple rules (common head, different bodies). For example, rule (2.5) provides the probability of $avoid$ given $waiting$ and $crossover\_dribble$ and rule (2.6) the probability of $avoid$ given $waiting$ and $running$, but we do not know this probability given all of the lower-level CEs. A combining rule could help us in computing such probabilities, without adding them explicitly.

2.5.2 Logic-based methods

Another line of research revolves around methods which employ a logical formalism to describe CE patterns, along with the necessary probabilistic extensions in order to handle uncertainty.

Markov Logic Networks

Since their first appearance Richardson and Domingos (2006), Markov Logic Networks (MLNs) have attracted increasing attention as a tool that can perform CE recognition under uncertainty. MLNs are undirected probabilistic graphical models which encode a set of weighted first-order logic formulas (for a comprehensive description of MLNs, see Domingos and Lowd (2009)). In first-order logic, a possible world (here meaning an assignment of truth values to all ground predicates) that violates even one formula is considered as having zero probability. With MLNs, possible worlds can have non-zero probability, even when violating some formulas, albeit a lower one than those without violations. The combination of a general but formal representation language together with a well-defined probability space constitutes an attractive feature of MLNs and has resulted in a significant number of research efforts.

For a simple example of how a formula in first-order logic is encoded as a MLN, consider a formula

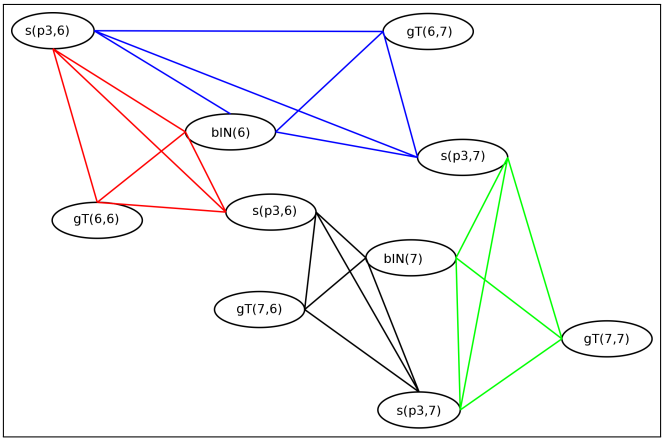

Figure 2.2: Markov Logic Network constructed for rule (2.7) and for part of the stream in Table 2.1. s=*shooting*, bIN=*ballInInet*, gT=*greaterThan*

in first-order logic about a player *scoring*:

$$\begin{aligned} score(X, T_2) \leftarrow & \\ & \forall X, T_1, T_2 \\ & shooting(X, T_1) \wedge \\ & ballInNet(T_2) \wedge \\ & greaterThan(T_2, T_1) \end{aligned} \tag{2.7}$$

Note that in first-order logic, it is not possible to directly perform numerical calculation and time (in-)equalities must be explicitly provided in the form of predicates, as above, with $greaterThan$. If we take into account only one player ($p3$) and only two timepoints (6,7) from the stream of Table 2.1, then the MLN corresponding to this formula would be the one shown in Figure 2.2, where each possible ground predicate is represented as a node and edges exist between nodes that appear together in a ground formula. Note that this is a direct and naive way of obtaining a ground MLN. Clever algorithms can prune unnecessary parts of the graph.

There is a substantial body of work on CER with MLNs Biswas et al. (2007); Helaoui et al. (2011); Tran and Davis (2008); Morariu and Davis (2011); Sadilek and Kautz (2012); Skarlatidis et al. (2011) Skarlatidis et al. (2015b); Song et al. (2013b,a); Kanaujia et al. (2014). All of these methods are concerned with human activity recognition, with input events derived mostly from video sources (less frequently from GPS or RFID traces). As a result, many of them have developed solutions that are domain dependent. Here we focus on those representative papers that are more closely related to CER, by providing a more generic way for handling events. E.g., in Morariu and Davis (2011) and Song et al. (2013b), where Allen's Interval Algebra Allen (1983b) is used and in Skarlatidis et al. (2011, 2015b), where a version of the Event Calculus Kowalski and Sergot (1986) is used. With the use of such formalisms, temporal constraints are not captured in the simplistic way implied by the $greaterThan$ predicate. Instead, built-in predicates about the temporal relations of events are provided, e.g., the $after$ relation in Allen's Interval Algebra, for indicating that an event succeeds in time another event. Interestingly, both the Interval Algebra and the Event Calculus represent time as intervals.

Contrary to automata-based solutions, MLNs focus on encoding probabilistic rules. As we have already mentioned, this allows both for incorporating background knowledge and for building hierarchies of CEs with correct probability propagation. On the other hand, they use the less intuitive weights

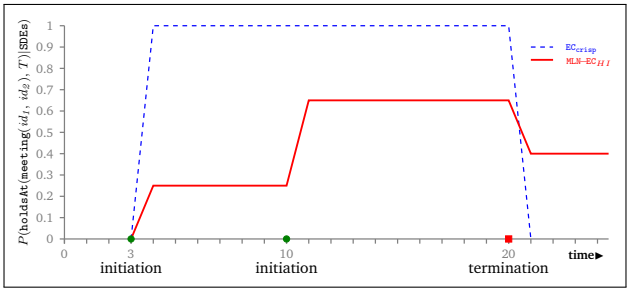

Figure 2.3: Probability increasing for every initiation point in MLN-EC, Skarlatidis et al. (2011, 2015b)

instead of probabilities, which indicate how strong a rule is compared to the others. While it might be possible for certain simple domains to manually define these weights, usually a learning phase is required to estimate their best values.

As far as data uncertainty is concerned, it is possible to include probabilistic SDEs as well, through formulas that "generate" them from the observed SDEs. Moreover, the flexibility of MLNs allows for more complex reasoning about SDEs. For example, in Tran and Davis (2008), besides handling noisy SDEs, missing SDEs may be inferred through rules about what must have happened for an event to have occurred.

A similar approach is proposed in Morariu and Davis (2011), where the Interval Algebra is employed and the most consistent sequence of CEs are determined, based on the observations of low-level classifiers. In order to avoid the combinatorial explosion of possible intervals, as well as to eliminate the existential quantifiers in CE definitions, a bottom-up process eliminates the unlikely event hypotheses. The elimination process is guided by the observations and the interval relations of the event definitions.

In Song et al. (2013b,a), MLNs are again combined with Allen's Interval Algebra. In this case, SDEs are not probabilistic, but a set of other domain-independent axioms is proposed. Abstraction axioms define hierarchies of events in which an instance of an event with a given type is also an instance of all abstractions of this type. Prediction axioms express that the occurrence of an event implies the occurrence of its parts. Constraint axioms ensure the integrity of the (temporal) relations among CE and its parts. Finally, abduction axioms allow CE to be inferred on the basis of their parts, i.e. inferring when some events are missing.

The work of Skarlatidis et al. Skarlatidis et al. (2011, 2015b) represents one of the first attempts to provide a general probabilistic framework for CER via MLNs. In order to establish such a framework, a version of the Event Calculus is used whose axioms are domain-independent. Combined with the probabilistic domain-dependent rules, inference can be performed regarding the time intervals during which events of interest (fluents in the terminology of the Event Calculus) hold. The problem of combinatorial explosion due to the multiple time-points that need to be taken into account is addressed by employing a discrete version of the Event Calculus, using only integer time-points and axioms that relate only successive time-points. For similar reasons, existential quantifiers are not allowed. Due to the law of inertia of the Event Calculus (something continues to hold unless explicitly terminated or initiated with a different value), this model increases the probability of an inferred event every time its corresponding rule is satisfied and decreases this probability whenever its terminating conditions are satisfied, as shown in Figure 2.3. The model requires the existence of such terminating conditions for every event.

Bayesian Networks

The work presented in Wasserkrug et al. (2008, 2012b,a) employs the technique of knowledge-based model construction (KBMC), whereby knowledge representation is separated from the inference process. Each event is assigned a probability, denoting how probable it is that the event occurred with specific values for its attributes. In turn, CE patterns are encoded in two levels, with a selection operation performing an initial filtering, mostly based on event type, followed by a pattern-detection schema for more complex operations, based on temporal and attribute constraints. The selection mechanism imposes certain independence properties on the network. CEs are conditioned only on selectable lower-level events (as determined by the selection operation), preventing the network from being cluttered with many dependency edges. This framework is not limited to representing only propositional or even first-order knowledge. It could potentially handle higher-order knowledge, since the pattern-matching step may, in principle, be defined in any kind of language. However, the system presented allows only predicates expressing temporal constraints on event timestamps and equality constraints on event attributes.

The calculation of probabilities for the CEs is done by a Bayesian network that is dynamically constructed upon each new event arrival. The nodes of the network correspond to SDEs and CEs. First, SDEs are added. Nodes for CEs are inserted only when a rule defining the CE is crisply satisfied, having as parents the events that triggered the rule, which might be SDEs or even other CEs, in case of hierarchical CE patterns. The attribute values of the inferred CEs are determined by mapping expressions associated with the corresponding rule, i.e. functions mapping attributes of the triggering events to attributes of the inferred event. In order to avoid the cost of exact inference, a form of sampling is followed, that bypasses the construction of the network by sampling directly according to the CE patterns.

A more recent effort extends the TESLA Cugola and Margara (2010) event specification language with probabilistic modelling, in order to handle the uncertainty both in input SDEs and in the CE patterns Cugola et al. (2014). The semantics of the TESLA language is formally specified by using a first-order language with temporal constraints that express the length of time intervals. At the input level, the system, called CEP2U, supports uncertainty regarding the occurrence of the SDEs, as well as the uncertainty regarding the content of the SDEs. SDEs are associated with probabilities that indicate a degree of confidence, while the attributes of a SDE are modelled as random variables with some measurement error. The probability distribution function of the measurement error is assumed to be known (e.g. Gaussian distribution). The method also models the uncertainty of CE patterns, by automatically building a Bayesian network for each rule. The probabilistic parameters of the network are manually estimated by domain experts.

Other methods based on Bayesian networks, which could be used for CE recognition include Bayesian logic programming (chapter 10 in Getoor and Taskar (2007)), relational Bayesian Networks Jaeger (1997) and relational dynamic Bayesian Networks Sanghai et al. (2005). Towards this direction, Dynamic Bayesian Networks have been extended using first-order logic Manfredotti (2009); Manfredotti et al. (2010). A tree structure is used, where each node corresponds to a first-order logic expression, e.g., a predicate representing a CE, and can be related to nodes of the same or previous time instances. Compared to their propositional counterparts, the extended Dynamic Bayesian Networks methods can compactly represent CE that involve various entities.

Comments on graphical models

Graphical models, such as Markov Logic Networks and Bayesian Networks, can provide a substantial degree of flexibility and they do not require restrictive independence assumptions to be made. Their power, however, lies in their ability to encode such assumptions and factorize the probability space for

more efficient inference. Moreover, they allow the adoption of discriminative modeling and therefore may avoid the explicit encoding of independence assumptions

On the other hand, this increased flexibility comes at a cost with respect to efficiency. In general, a rule which references certain random variables implies that, before inference can begin, the cartesian product of all the values of these variables needs to be taken into account. For human activity recognition, one may assume that the number of persons involved in a scene is relatively limited. However, this is not the case for all domains. For a fraud detection scenario, involving transactions with credit cards, a CER system may receive thousands of transactions per second, most of them having different card IDs. The demands of event recognition exacerbate this problem, since time is a crucial component in these cases. The possible combinations of time points with the other random variables can quickly lead to intractable models. All of the papers discussed in this section employ low-arity (or even 0-arity) predicates, whose arguments have small domain sizes, except for that of time. In order to reduce the unavoidable complexity introduced by the existence of time, they develop special techniques, such as the bottom-up technique in Morariu and Davis (2011).

With respect to probabilistic SDEs, although they can be incorporated into graphical models, correctly encoding their dependencies can be far from obvious, especially with MLNs. Assume we want to assign an occurrence probability of 80% to the $close$ SDE of rules (2.3)–(2.4). With MLNs, we could replace rule (2.3) with an equivalence rule, such as:

$$1.39\ \forall X, Y, T\ close(X, Y, T) \leftrightarrow close\_m(X, Y, T) \tag{2.8}$$

with the appropriate weight (log-odds of occurrence and non-occurrence probabilities), where $close$ are the observed SDEs and $close\_m$ the inferred probabilistic events to be used in other rules, such as (2.2). It would not be sufficient to directly express rules (2.4) and (2.8) in first-order logic and use them to construct an MLN, since the dependencies introduced would mean that the marginal probability of the $close$ SDE could be affected by the probability of the $close\_m(X, Y, T_{previous})$ predicate. Bayesian Networks could be used to avoid such problems, due to their directionality. On the other hand, MLNs can be trained as discriminative models, which can result in a simpler structure that does not require determining all the dependencies among the evidence variables.

Probabilistic Logics

Similar to MLNs, the Probabilistic Event Logic (PEL) Brendel et al. (2011); Selman et al. (2011) method has been used to define a log-linear model from a set of weighted formulas, but the formulas are represented in Event Logic, a formalism for defining interval-based events Siskind (2001). Each formula defines a soft constraint over some events, using interval relations that are represented by a specialised data structure (*spanning intervals*). The method performs inference via a local-search algorithm (based on MaxWalkSAT of Kautz et al. (1997)), but, by using *spanning intervals*, it avoids grounding all possible time intervals.

Another probabilistic activity description language for expressing CE on top of an image processing suite is proposed in Albanese et al. (2010). This method merges logic-based formulation with probabilistic modeling and can handle both instantaneous events and events that span over intervals. The input SDEs can be either Boolean or probabilistic. CEs are defined by users in first-order logic where the dependencies between SDEs are modeled by triangular norms Fagin (1996), i.e. binary operations that can specify more general probabilistic relationships, besides independence and exclusivity.

In Skarlatidis et al. (2013) the Event Calculus was again used, similarly to Skarlatidis et al. (2011), but this time the focus was on probabilistic SDEs rather than probabilistic rules. The ProbLog logic programming framework was used Kimmig et al. (2011), which allows for assigning probabilities to SDEs

and can compute the success probability of a query by summing the probabilities of all the subprograms that entail it. The method exhibited improved accuracy performance with respect to crisp versions of the Event Calculus on a dataset of video recognition.

2.5.3 Petri Nets

In order to address the issues of concurrency and synchronization, a probabilistic extension to Petri-Nets has been proposed in Albanese et al. (2008), for recognizing CEs that represent human activities. A Petri-Net expresses a CE and is formed by SDEs that are connected with constraints (temporal durations and forbidden actions). The transition between a pair of subsequent SDEs is associated with a probability value. Given a sequence of SDEs, the method can identify segments of the sequence in which a CE occurs with probability above a specified threshold or infer the most likely CE in the sequence.

A stochastic variant of Petri-Nets that models the uncertainty of input SDEs is proposed in Lavee et al. (2013), an issue not addressed in Albanese et al. (2008). Specifically, SDEs are recognized with some certainty, using lower level classification algorithms. CEs are represented by Petri-Nets, in terms of SDEs that are associated with certainties and temporal constraints. However, the rules themselves cannot be probabilistic.

One limitation of the approaches that use Petri Nets is the lack of a mechanism for modelling a domain in a truly relational manner, i.e. by allowing relations to be defined between attributes of events. These methods treat events as 0-arity predicates, related only through temporal constraints, as implied by the structure of the Petri Net, i.e. events are reified, in the terminology of logic. As is the case with automata too, Petri Nets tend to make a significant number of independence assumptions. The domain on which they have been tested is that of human activity recognition, in which the sequential nature of activities allows for the adoption of first-order Markov models. As far as inference is concerned, both MAP and marginal are possible and optimization techniques, such as confidence thresholds and approximate inference have been employed.

2.5.4 Context-Free Grammars

A number of research efforts have focused on syntactic approaches to CE recognition. These approaches typically convert a stream of input SDEs to a stream of symbols upon which certain user-defined rules may be applied. Rules are defined via a stochastic context-free grammar Stolcke (1995) in order to take uncertainty into account. A two-step approach along these lines is proposed in Ivanov and Bobick (2000). Low-level detectors, based on Hidden Markov Models, are used to generate a symbol stream which is then fed into a parser constructed from a stochastic context-free grammar. This parser in turn employs stochastic production rules in order to determine the probability of a high-level event. Similar approaches may be found in Moore and Essa (2002) and in Minnen et al. (2003). Note though that the latter adds context-sensitive symbols to the grammar, i.e., symbols/events may have arguments.

Recognizing that such syntactic approaches are unable to handle concurrency, a hierarchical method is proposed in Ryoo and Aggarwal (2009), that combines a syntax for representing the CE patterns of Ryoo and Aggarwal (2006) with probabilistic recognition. A syntax similar to that of context-free grammars is used for describing CEs, but the actual recognition is treated as a constraint satisfaction problem. The method aims to probabilistically detect the time intervals in which CEs occur. Input SDEs are associated with probabilities, indicating a degree of belief. Based on a context-free grammar representation scheme, CE patterns are expressed in terms of other events (SDEs or CEs) and form a hierarchical model. Furthermore, events are related with logical, spatial and temporal interval constraints Allen (1983a). In situations where the input stream is incomplete, the method generates the missing SDEs with low confi-

Language Expressivity										
Paper	σ	π	$\vee$	$\neg$	;	*	**W**	**H.**	**T.M.**	**Background Knowledge**
Automata										
SASE+	✓	✓			✓	✓	✓		Points,Implicit.	
Lahar	✓				✓	✓			Points,Implicit.	
Chuanfei et al.					✓	✓	✓		Points,Implicit.	
SASE+ AIG	✓	✓			✓	✓	✓		Points,Implicit.	
Albanese et al.					✓	✓			Points,Implicit.	
SASE+ opt. AIG	✓		✓	✓	✓		✓	✓	Points,Implicit.	
SASE++	✓	✓	✓	✓	✓	✓	✓		Points,Implicit.	
Logic-based										
KBMC	✓	✓	✓		✓			✓	Points,Explicit.	
CEP2U	✓	✓		✓	✓		✓	✓	Points,Implicit.	
PEL			✓	✓	✓			✓	Intervals,Implicit. Allen's Interval Algebra.	✓
PADS	✓		✓	✓	✓				Intervals,Explicit.	
MLN-Allen	✓	✓	✓	✓	✓		✓	✓	Intervals,Explicit. Allen's Interval Algebra.	✓
MLN-Event Calculus	✓	✓	✓	✓				✓	Points,Explicit. Event Calculus.	✓
MLN-hier.			✓	✓	✓			✓	Intervals,Explicit. Allen's Interval Algebra.	✓
Prob-Event Calculus	✓	✓	✓	✓				✓	Points,Explicit. Event Calculus.	✓
Petri Nets										
PPN	✓		✓	✓	✓	✓			Points,Implicit.	
PFPN			✓		✓				Points,Implicit.	
Context-Free Grammars										
Ivanov et al.			✓		✓	✓		✓	Intervals,Implicit.	
Ryoo et al.	✓		✓	✓	✓			✓	Intervals,Implicit. Allen's Interval Algebra.	
Paper	σ	π	$\vee$	$\neg$	;	*	**W**	**H.**	**T.M.**	**Background Knowledge**

Table 2.2: Expressive capabilities of CER systems. σ: selection, π: production, $\vee$: disjunction, $\neg$: negation,
;: sequence, *: iteration, W: windowing, H: Hierarchies, T.M.: Temporal Model

Probabilistic Expressivity					
Paper	**Model**	**Independence assumptions**	**Data uncertainty**	**Patterns**	**H. C.**
Automata					
SASE+		All events independent.	Occurrence		
Lahar		1st-order Markov for SDEs. Different streams independent.	Occurrence/ Attributes		
Chuanfei et al.		1st-order Markov with extensions.	Occurrence/ Attributes		
SASE+ AIG		SDEs independent.	Occurrence/ Attributes		
Albanese et al.	Patterns modelled as stochastic activities/processes, similar to Markov chains.	1st-order Markov within activity. Different activities independent.		✓	
SASE+ opt. AIG		SDEs independent or Markovian. Different streams independent.	Occurrence		
SASE++	Probability distribution on time attribute.	SDEs independent.	Occurrence		
Logic-based					
KBMC	Bayesian Networks.	SDEs independent.	Occurrence/ Attributes	✓	
CEP2U	Bayesian Networks.	Event attributes independent. SDEs independent. CEs dependent only on events immediately below in hierarchy.	Occurrence/ Attributes	✓	
PEL	Weights, as in Conditional Random Fields.	None		✓	✓
PADS	Probabilities assigned to predicates for object equality.	None. Depends on t-norm.	Occurrence		
MLN-Allen	Markov Logic Networks. Bottom-up hypothesis generation.	None.	Occurrence	✓	✓
MLN-EC	Markov Logic Networks.	None.		✓	✓
MLN-hier.	Markov Logic Networks.	None		✓	✓
Prob-EC	Probabilistic Logic Programming.	SDEs independent.	Occurrence		
Petri Nets					
PPN	Hard constraints as forbidden actions. As in Albanese et al. (2011), activities resemble stochastic processes.	Conditioned on previous event in pattern.		✓	✓
PFPN		1st-order Markov. SDEs independent.	Occurrence		
Context-Free Grammars					
Ivanov et al.	Probability space defined over proofs.	Rules conditionally independent.	Occurrence	✓	
Ryoo et al.		Conditional independence of SDEs.	Occurrence		
Paper	**Model**	**Independence assumptions**	**Data uncertainty**	**Patterns**	**H. C.**

Table 2.3: Expressive power of CER systems with respect to their probabilistic properties. H.C.: Hard Constraints

Inference					
Paper	**Type**	**Confidence Thresholds**	**Approx.**	**Distrib.**	**Performance**
Automata					
SASE+	Marginal	✓			0.8-1.1 K events/sec with $Kleene+$.
Lahar	Marginal	✓			$>$ 10 points increase in accuracy. 100K events/sec for Extended Regular Queries.
Chuanfei et al.	Marginal	✓			4-8K events/sec for patterns of length 6 down to 2.
SASE+ AIG	Marginal	✓			1000K events/sec, almost constant for varying window size. 1000K-100K events/sec for experiments with 1 up to 10 alternatives of a SDE.
Albanese et al.	Marginal and MAP	✓		✓	Running time linear in video length. Better F-measure with the extensions in Molinaro et al. (2014). Parallel version reached 335K events/sec with 162 computing nodes.
SASE+ opt. AIG	Marginal	✓		✓	8K-13K events/sec for 2-6 nodes.
SASE++	Marginal	✓			Reduction from exponential to close-linear cost w.r.t to selectivity / window size.
Logic-based					
KBMC	Marginal		✓		CEs within desired confidence interval. Sub-linear decay of event rate w.r.t possible worlds.
CEP2U	Marginal	✓			50% overhead w.r.t deterministic case.
PEL	MAP		✓		Smooth accuracy degradation when noise in time intervals of SDEs added. Relative robustness against false positives/negatives.
PADS	Marginal	✓			Running time at most linear in the number of atoms. Better precision/recall than Hidden Markov Models/ Dynamic Bayesian Networks, higher computation time.
MLN-Allen	Marginal				F-measure $>$ 70% for varying window sizes.
MLN-EC	Marginal				Increased precision, slight decrease in recall, compared to deterministic solution.
MLN-hier.	MAP				
Prob-EC	Marginal				Improved F-measure w.r.t. crisp version.
Petri Nets					
PPN	Marginal and MAP	✓			$\sim$ 3 seconds to process videos $\sim$ with 60 different SDE types.
PFPN	Marginal		✓		Increased true positive rate, compared to deterministic solution. Slight increase in false positive rate.
Context-Free Grammars					
Ivanov et al.	Marginal	✓			
Ryoo et al.	Marginal	✓			Increased accuracy when noisy SDEs present.
Paper	**Type**	**Confidence Thresholds**	**Approx.**	**Distrib.**	**Performance**

Table 2.4: Inference capabilities of probabilistic CER systems

dence. The probability of a CE is calculated by exploiting the dependency information between the CE and its sub-events, as encoded in the hierarchy tree. This method for calculating probabilities is similar to that of Bayesian networks, the main difference being that siblings are not assumed to be conditionally independent. When the calculated probability of the CE is above a specified threshold, it is considered as recognized.

The SDEs of Ivanov and Bobick (2000) (the terminal symbols) are represented as 0-arity, hence no relations may be defined on attributes (the addition of sensitivity in Minnen et al. (2003) provides a more flexible approach). Since no event attributes are allowed, it is not possible to define probabilities on attributes either. Moreover, when defining a (production) rule, all the possible sub-scenarios (expansions) must be explicitly stated, with probability values that sum to 1. For example, a rule for detecting the *avoid* event, as in rule (2.5), cannot be "simply" written, as:

$$\begin{aligned} 0.9 :: & Avoid\_Player1\_Player2 \quad \rightarrow \\ & Waiting\_Player1\_Player2, \\ & Crossover\_Dribble\_Player2 \end{aligned} \tag{2.9}$$

All the possible scenarios for $Avoid$ need to be explicitly provided In this example, rule (2.6) should be added as:

$$\begin{aligned} 0.1 :: & Avoid\_Player1\_Player2 \quad \rightarrow \\ & Waiting\_Player1\_Player2, \\ & Running\_Player2 \end{aligned} \tag{2.10}$$

However, note that the two scenarios are considered as mutually exclusive, with a total sum probability of 1. Note also that events are again reified, as is the case with Petri Nets.

2.6 Discussion

Table 2.2 summarizes the expressive power of the presented CER systems. Its first columns correspond to the list of operators presented in Section 2.4.1. The other columns assess various aspects of the functionality supported by each system. These are:

- Hierarchies: A column to indicate whether a system supports event hierarchies, i.e. the ability to define CEs at various levels and reuse those intermediate inferred events in order to infer other higher-level events.
- Temporal Model: The temporal model used. Events may be represented by timepoints (P) or intervals (In). Moreover, the time attribute might be explicitly included in the constraints (Ex) (e.g. $(T_2 > T1) \wedge (T_2 - T1 > 100)$) or temporal constraints may be defined by referring implicitly to time in the rules (Im) (e.g. the $Sequence$ operator implicitly defines $T_2 > T1$).
- Background Knowledge: Does the system support background knowledge, besides what is encoded in the CE patterns?

In Table 2.3 we present the probabilistic properties of each method:

- Model: The probabilistic model used.
- Independence assumptions: What are the independence/dependence assumptions made?
- Data: Does the system support data uncertainty? If yes, is it only about the *occurrence* of events or *both* about the occurrence and the event attributes?

Approach	Strengths	Weaknesses
Automata	*Iteration*, *Windowing*, formal Event Algebra.	Limited support for event hierarchies. No background knowledge. Implicit time representation (hence no explicit constraints on time attribute).
	Data uncertainty, both with respect to occurrence of events and event attributes.	Limited or no support for rule uncertainty. Too many independence assumptions. No hard constraints.
	Existence of confidence thresholds. High throughput values.	Throughput figures come from experiments with simplistic event patterns.
Logic	Complex temporal patterns, with explicit time constraints. Event hierarchies. Background knowledge. Usually provide a formal Event Algebra.	No *Iteration*. Limited support for *Windowing*.
	Rule uncertainty. Limited independence assumptions. Hard constraints possible.	Harder (but not impossible) to express data uncertainty. Often training required (experts cannot "simply" assign probabilities to rules).
	MAP and approximate inference.	Under-performance with respect to throughput.
Petri Nets	Concurrency and synchronization.	Not truly relational, event instances reified. No *Windowing*, hierarchies or background knowledge. Implicit time representation.
	Have been shown to support both data and rule uncertainty (but not both in the same model).	Strict independence assumptions. Have not been shown to support continuous domains.
	Can perform both MAP and Marginal inference. Confidence thresholds and approximate inference possible.	Low (or unknown) throughput.
Grammars	Very easy to model hierarchies and *Iteration*. Recursive patterns.	Not truly relational (unless context-sensitive grammars are used). No *Negation*. Implicit time representation. No background knowledge.
	Both data and rule uncertainty.	No rich methodology for efficient probabilistic inference.
	Confidence thresholds.	Unknown performance for throughput.

Table 2.5: Strengths and weaknesses of the reviewed probabilistic CER approaches

- Patterns: Is there support for uncertain patterns?
- Hard constraints: Can the user specify rules that should not be violated?

Table 2.4 presents some of the inference capabilities of the presented systems:

- Type: Can the system perform *Marginal* inference, *MAP* inference or *both*?
- Confidence Thresholds: Can the user define confidence thresholds above which a CE is accepted?
- Approximate: Does the system support techniques for approximate inference?
- Distribution: Is there a distributed version of the proposed solution?
- Performance: Remarks about performance with respect to throughput, latency and accuracy. Note that such information is not always available.

Our review of probabilistic CER systems identified a number of strengths and limitations for the proposed approaches. We summarize our conclusions in Table 2.5. As a note of caution though, we have to mention that, when a weakness is reported, this does not mean that the corresponding method cannot in general support a feature (as might be the case, for example, for $Iteration$ in first-order logic), but that the presented methods have not incorporated it, although it might be possible (for example, the absence of hierarchies in automata-based methods).

The current systems for probabilistic CER need to deal with a trade-off between language expressivity, probabilistic expressivity and complexity. High throughput figures can be achieved only when simple patterns and probabilistic models are used. Achieving the same figures with more complex patterns/models so that online, real-time inference is possible still remains a challenge and very few approaches have explored distributed solutions. Moreover, only rarely do these systems attempt to learn the weights or structure of patterns and instead rely on experts to define them. This might be sufficient for simple patterns, but would not scale well in case multiple, complex rules with complicated dependencies are required.

3

Machine Learning: Approach

3.1 Introduction

Both fraud detection and traffic management applications depend upon multiple layered and distributed information systems. As time evolves, such systems share, collect and process data in various structured and unstructured digital formats. When such information is aggregated and correlated, it might become a significant source of knowledge for an organization, allowing for the representation of important activities. These pieces of information, together with their temporal occurrence, can be represented by *events*.

Subsequently, automatic recognition and forecasting of significant events can be performed by systems that employ Complex Event Processing (CEP) techniques. The aim of a CEP system is to recognize *composite events* (CEs) of interest, based on input streams of time-stamped symbols, that is, *simple, derived events* (SDEs). CEs are defined as relational structures over other sub-events, either CEs or SDEs. Such CE definitions have the form of rules, usually expressed in a formal language, that capture the knowledge of domain experts. Due to the dynamic nature of the aforementioned use cases, the CE definitions may need to be refined or it might be required that the current knowledge based is enhanced with new definitions. Manual creation of event definitions is a tedious and cumbersome process. Thus, machine learning techniques to automatically derive event definitions are required.

Furthermore, uncertainty is an unavoidable aspect of real-world event recognition applications and it appears to be a consequence of several factors (Shet et al. 2007; Artikis et al. 2010a; Etzion and Niblett 2010b, Section 11.2; Gal et al. 2011; Skarlatidis 2014). Under environments characterized by uncertainty, the performance of an event recognition system may be seriously compromised. There are three types of uncertainty that might appear in an event recognition application: (a) Erroneous input SDEs, (b) Incomplete input streams and/or (c) Imperfect event definitions.

To overcome the issues, we are developing probabilistic inference and structure learning algorithms that operate under noisy environments and scale to very large datasets. Specifically, we combine a well-defined temporal logic formalism with statistical relational modeling and learning methods. We present an approach on scalable incremental learning of event definitions from large amounts of data.

In Section 3.2 we begin by briefly presenting background work for both batch and online parameter learning. Then, in Section 3.3 we present related work for structure learning and outline the limitations of these approaches. Finally, in Section 3.4 we present our developed algorithm OSLα.

3.2 Background on Parameter Learning

The weights of the soft-constrained clauses in Markov Logic Networks (MLNs) can be estimated from training data, using supervised learning techniques. As mentioned in D3.1 (Skarlatidis et al., 2014), in many applications (including event recognition), the actual goal is to maximize an alternative performance metric to conditional log-likelihood (CLL) such as classification accuracy or F-measure. Thus, an alternative to optimization to CLL is max-margin training which constitutes an approach competitive to discriminative training and also has the advantage that it can be adapted to maximize a variety of performance metrics beyond classification accuracy (Joachims, 2005). Furthermore, max-margin methods have been successfully applied to structure prediction (Taskar et al., 2003; Tsochantaridis et al., 2005) in order to learn parameters that maximize the margin between the probability of the correct assignment and that of the other assignments. Instead of optimizing the CLL, max margin methods maximize the following ratio:

$$\frac{P(Y{=}\mathbf{y}|X{=}\mathbf{x},\mathbf{w})}{P(Y{=}\hat{\mathbf{y}}|X{=}\mathbf{x},\mathbf{w})}$$

The above equation measures the ratio between the probability of correct truth assignment $\mathbf{y}$ of CEs and the closest competing incorrect truth assignment $\hat{\mathbf{y}}{=}\text{argmax}_{\bar{\mathbf{y}}\in\mathbf{Y}\setminus\mathbf{y}}P(Y{=}\bar{\mathbf{y}}|X{=}\mathbf{x})$, also known as separation oracle. We employ the method of Huynh and Mooney (2009) which formulates the max-margin problem as a 1-slack structural support vector machine (SVM) using a cutting-plane algorithm proposed by Joachims et al. (2009). Specifically, structural SVMs predict structured outputs instead of discrete labels or real values. In particular, they describe the problem of learning a function $h : \mathcal{X} \mapsto \mathcal{Y}$, where $\mathcal{X}$ is the space of input examples, and $\mathcal{Y}$ is the space of multivariate and structured outputs from the set of training examples $S = ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)) \in (\mathcal{X} \times \mathcal{Y})^n$.

The goal is to find a function h that has low prediction error. This can be accomplished by learning a discriminant function $f : \mathcal{X} \times \mathcal{Y} \mapsto \mathcal{R}$ and maximize f over all $\mathbf{y} \in \mathcal{Y}$ for a given input $\mathbf{x}$ in order to get a classifier of the form:

$$h_w(\mathbf{x}) = \underset{\mathbf{y}\in\mathcal{Y}}{\text{argmax}} f_w(\mathbf{x}, \mathbf{y})$$

The discriminant function $f_w(\mathbf{x}, \mathbf{y}) = \mathbf{w}^T\mathbf{\Psi}(\mathbf{x}, \mathbf{y})$ is linear in the space of features, where $\mathbf{w} \in \mathcal{R}^N$ is a parameter vector and $\mathbf{\Psi}(\mathbf{x}, \mathbf{y})$ is a feature vector relating an input $\mathbf{x}$ and output $\mathbf{y}$. The features need to be designed for a given problem so that they capture the dependency structure of $\mathbf{y}$ and $\mathbf{x}$ and the relations among the outputs $\mathbf{y}$. In our case $\mathbf{\Psi}(\mathbf{x}, \mathbf{y}) = \mathbf{n}(\mathbf{x}, \mathbf{y})$ is the number of satisfied groundings of $\mathbf{x}$ and $\mathbf{y}$. Therefore, the goal is to find a parameter vector $\mathbf{w}$ that maximizes the margin by employing linear programming techniques described in Huynh and Mooney (2009).

Batch training algorithms must repeatedly run inference over all training examples in each iteration. This becomes computationally expensive and eventually infeasible for very large datasets with thousands of training examples, which may not even fit in the main memory. To overcome this problem, we employ an online learning strategy where the learner sequentially processes a set of examples at each step in the form of micro-batches. The size of these batches can be adjusted.

In particular, we employ an online max-margin method proposed by Huynh and Mooney (2011a) based on the Coordinate-Dual-Ascent update rule (CDA). The algorithm is derived from the primal-dual framework for strongly convex loss functions (Hazan et al., 2006), which is a framework for deriving online algorithms that have low regret. CDA can use various loss functions to guide the optimization. We employ a prediction-based loss which measures the difference between the predicted possible world

and the ground-truth one. The update rule in terms of the weight vector for each step t is the following:

$$\mathbf{w}_{t+1} = \frac{t-1}{t}\mathbf{w}_t + \frac{1}{\sigma t}\Delta\phi_t \tag{3.1}$$

where σ is a non-negative constant and ϕ_t is a feature vector representing the number of satisfied groundings in $\mathbf{x}$ and $\mathbf{y}$ as in the batch version of max-margin training. Note that the learning rate of the update rule is controlled by the loss suffered at each step. At the beginning, when the model has low predictive quality, CDA aggressively updates the model based on the loss suffered at each step. Later, when the model improves, it is updated less aggressively.

Although CDA is quite fast, it may lack in accuracy. Moreover, Lee et al. (2006) states that parameter learning algorithms used by the structure learning procedure, which may introduce a lot of new features and many of which may not be useful, perform better when they use L1-regularization. Therefore, L1-regularization, which has the tendency to force parameters to zero and thus leads us to sparser models, is preferred over other Lp norm regularization methods. For this reason we also employ another algorithm for online optimization used by Huynh and Mooney (2011b) in order to perform structure learning more efficiently.

AdaGrad, proposed by Duchi et al. (2011), is a state-of-the-art online method, which is an L1-regularized adaptive subgradient algorithm based on composite mirror-descent updates. AdaGrad belongs to a family of subgradient methods that dynamically incorporate knowledge of the geometry of the data observed in earlier steps to perform more informative gradient-based learning. Informally, the algorithm gives frequently occurring features very low learning rates and infrequent features high learning rates, where the intuition is that each time an infrequent feature is seen, the learner should "take notice". Thus, the adaptation facilitates finding and identifying very predictive but comparatively rare features. It is often the case that infrequently occurring features are highly informative and discriminative. Indeed, in many applications of online and stochastic learning, the input instances are of very high dimensionality, yet within any particular instance only a few features are non-zero. AdaGrad updates the weight vector at each step t as follows:

$$\mathbf{w}_{t+1,i} = \text{sign}\Big(\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\Big)\Big[\Big|\mathbf{w}_{t,i} - \frac{\eta}{H_{t,ii}}\mathbf{g}_{t,i}\Big| - \frac{\lambda\eta}{H_{t,ii}}\Big]_+ \tag{3.2}$$

where λ is the regularization parameter used to tackle overfitting, η is the learning rate, $\mathbf{w}_{t,i}$ is the i-th component of the weight vector at step t and $[a]_+$ denotes a truncated function at 0, i.e. $[a]_+ = \max(a, 0)$. The function sign returns the sign of the operation inside the brackets. The subgradient $\mathbf{g}_t$ is computed, at each step t, over the loss function which guides the optimization process, as follows:

$$\mathbf{g}_{PL} = \mathbf{n}_C(\mathbf{x}_t, \mathbf{y}_t^{PL}) - \mathbf{n}_C(\mathbf{x}_t, \mathbf{y}_t) = \Delta\mathbf{n}_C \tag{3.3}$$

where $\mathbf{n}_C$ is the number of true groundings for the clause C and $\mathbf{y}_t^{PL}$ is the predicted possible world. The subgradient of the loss function represents the difference between the number of satisfied groundings computed over the predicted possible world and the ground-truth one. Regarding the loss function, we use the prediction-based loss function, also used by Huynh and Mooney (2011b) which is a simpler variant of the max-margin loss (Huynh and Mooney, 2009). Furthermore, the subgradient is also required to compute $H_{t,ii}$ as follows,

$$H_{t,ii} = \delta + ||\mathbf{g}_{1:t,i}||_2 = \delta + \sqrt{\sum_{j=1}^{t}\mathbf{g}_{j,i}^2} \tag{3.4}$$

where $\mathbf{H}$ is a diagonal matrix and δ is the default value of the matrix. Note that AdaGrad assigns a different step size $\frac{\eta}{H_{t,ii}}$ for each component of the weight vector. Thus, AdaGrad needs to retain the sum of squared subgradients of each component. From the update rule we can see that if a clause is not relevant to the current example the AdaGrad discounts its weight by $\frac{\lambda\eta}{H_{t,ii}}$. Thus, irrelevant clauses will be zeroed out in the long run.

3.3 Related Work

The task of structure learning is to discover the dependency structure of the model as well as estimate the parameters of these dependencies given a set of training examples $\mathcal{D}$. The training set usually consists of a single interconnected example containing many ground instances of observed SDEs as well as unobserved CEs (query predicates). Nearly all approaches developed for structure learning perform a heuristic search through the space of possible structures, known as the hypothesis space $\mathcal{H}$, in order to avoid exhaustive search, which is combinatorially explosive for complex models. Typically, given a particular scoring function $\mathcal{S}(h, \mathcal{D})$ for $h \in \mathcal{H}$ the task is reduced to finding a hypothesis $h^* \in \mathcal{H}$ that maximizes the score, i.e. $h^* = \text{argmax}_{h \in \mathcal{H}} \mathcal{S}(h, \mathcal{D})$.

In MLNs the structure is a set of weighted formulas in conjunctive normal form. In principle, the structure can be learned or revised by using approaches for learning graphical models (Pietra et al., 1997; Heckerman, 1999; McCallum, 2012) as well as Inductive Logic Programming (ILP) techniques (Quinlan, 1990; De Raedt and Dehaspe, 1997; Srinivasan, 2004). However, since an MLN represents a probability distribution over possible worlds, much better results are obtained by using evaluation functions based on likelihood (e.g. pseudo-likelihood), rather than typical ILP ones like accuracy and coverage (Kok and Domingos, 2005). Log-likelihood or conditional log-likehood are probably the best evaluation functions, albeit particularly expensive to compute.

In this section we outline important batch approaches that have been developed in order to efficiently learn and revise the MLN structure, starting either from an existing knowledge base or an empty one. These methods have proven to be beneficial in many real-world applications in citation analysis, web mining, natural language processing, robotics, bioinformatics, computer games as well as activity recognition.

Top-down structure learning Top-down structure learning as proposed by Kok and Domingos (2005) learn or revise the MLN structure in an iterative fashion. The initial structure can be an empty network or an existing KB. At each step, the algorithm searches for the best clause to add to the model. Searching can be performed using one of two possible strategies. The first one, *beam search*, keeps the best k clause candidates at each step of the search. The second one, *shortest-first search*, tries to find the best clauses of length i before it moves on to clauses of length $i+1$. Candidate clauses are formed by adding each predicate (negated or not) to each current clause, using all possible combinations of variables, subject to the constraint that at least one variable in the new predicate must appear in the current clause. Candidate clauses are scored using the weighted pseudo log-likelihood measure, an adaptation of the pseudo log-likelihood that weights the pseudo-likelihood of each grounded atom by 1 over the number of groundings of its predicate, in order to prevent predicates with larger arity from dominating the expression. The procedure follows a blind generate-and-test strategy in which many potential changes to an existing model are systematically generated independently of the training data, and then tested for empirical adequacy. For complex models such as MLNs, the space of potential revisions is combinatorially explosive and such a search procedure can become difficult to control, resulting in convergence to suboptimal local maxima.

Iterative Local Search One way of addressing the potential shortcomings of greedy structure selection is by using iterative local search techniques (Loureno et al., 2003), that explores the space of structures through a biased sampling of the set of local optima found by a local search procedure. They focus the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for the optimization engine. These techniques alternate between two types of search steps: (a) either moving towards a locally optimal solution using a given evaluation function, or (b) perturbing the current solution in order to escape from local optima. The algorithm proposed by Biba et al. (2008) uses the above technique to avoid local maxima when learning MLNs in a discriminative setting, where the focus is on predicting a specific target predicate given the evidence on all other predicates.

Bottom-up structure learning Another alternative is using algorithms developed to restrict the hypothesis space $\mathcal{H}$, typically by performing a pre-processing step that, roughly speaking, discovers more promising regions of the space. Bottom-Up Structure Learning (BUSL) introduced by Mihalkova and Mooney (2007) is based on the observation that, once an MLN is instantiated into a Markov network (through the grounding procedure), the instantiations of each clause of the MLN define a set of identically structured cliques in the Markov network. BUSL inverts this process of instantiation and constrains the search space by inducing lifted templates for such cliques in order to learn a so-called *Markov network template*. The template network is composed of *template nodes*, conjunctions of one or more literals that serve as building blocks for creating clauses. Template nodes are constructed by looking for groups of constant-sharing ground literals that are true in the data and abstracting them by substituting variables for constants. Thus, these template nodes could also be viewed as portions of clauses that have true groundings in the data. To understand why conjunctions of literals with true groundings are good candidates for clause components, consider the special case of a definite clause $L_1 \wedge \cdots \wedge L_n \Rightarrow P$. If the conjoined literals in the body have no true groundings, then the clause is always trivially satisfied. Therefore, true conjunctions will be most useful for building effective clauses. To search for these, BUSL generates clause candidates by focusing on each maximal clique in turn and producing all possible clauses consistent with it. The candidates are then evaluated using the weighted pseudo log-likelihood score. BUSL restricts its search for clauses only to those candidates whose literals correspond to template nodes that form a clique in the template. It also makes a number of additional restrictions on the search in order to decrease the number of free variables in a clause, thus decreasing the size of the ground MLN during inference, and further reducing the search space. Therefore, BUSL typically restricts the search to very short paths, creating short clauses from them and greedily joining them into longer ones. Although this approach is faster and yields more accurate models than the top-down approach, it may still converge to a suboptimal local maxima.

Discriminative heuristic structure learning An approach somewhat similar to the principle underlying the BUSL algorithm was proposed by Dinh et al. (2010). Heuristic Method for Discriminative Structure Learning employs a heuristic method in order to discriminatively learn the structure of MLNs. It consists of three main steps. First, for each true ground query atom, applies a heuristic technique to build a set of variable literals, called chain. To achieve that is adds to the set each ground atom in the training example $\mathcal{D}$ connected to the ground query atom through its arguments and variabilize it. Then transforms the learning dataset to a boolean table, having ground query atoms as rows and variable literals as columns, representing their connections. Second, by applying the Grow-Shrink Markov Network (Bromberg F, 2009) algorithm to these boolean tables, it extracts a set of template clauses. A template clause is a disjunction of positive variable literals. Finally, candidate clauses are built from the template clauses to be added into the MLN. The score of the weighted MLN is then measured by computing either its conditional log-likelihood or weighted pseudo log-likelihood. The procedure follows a similar

approach to the BUSL algorithm. Both of them consist of three mains steps: Transforming the relational dataset into template clauses, building candidate clauses using these templates and putting clauses into the MLN. Although the quality of the boolean tables constructed yields higher accuracy, the greedy strategy delivers higher running times.

Moralized Bayes Nets An alternative approach is searching for structures of increasing complexity at each stage of the procedure; using the structures found at the previous stage to constrain the search space. Such a strategy was employed by Khosravi et al. (2010) for learning MLN structure in domains that contain many descriptive attributes, namely predicates having very large arity. Their approach, which is similar to the technique employed to constrain the search space in Probabilistic Relational Models (Friedman et al., 1999), distinguishes between two types of tables – attribute tables that describe a single entity type, and relationship tables that describe relationships between entities. The algorithm, called MBN (Moralized Bayes Net), proceeds in three stages. In the first stage, dependencies local to attribute tables are learned. In the second stage, dependencies over a join of an attribute table and a relationship table are learned, but the search space is constrained by requiring that all dependencies local to the attribute table found in the first stage remain the same. Finally, in the third stage, dependencies over a join of two relationship tables, joined with relevant attribute tables are learned and the search space is similarly constrained. Although the goal of MBN is to learn an undirected model, dependencies are learned using a Bayesian network learner and then the directed structures are converted to undirected ones by using moralization (Cowell et al., 2007). The advantage of this approach is that structure learning in directed models is significantly faster than structure learning in undirected models due to the decomposability of the score, which allows it to be updated locally, only in parts of the structure that have been modified, making scoring of candidate structures more efficient. On the other hand, when scaling to larger table joins, the algorithm becomes computationally intensive.

Hypergraph Lifting Another MLN learner that is based on constraining the search space is the Learning via Hypergraph Lifting (LHL) algorithm introduced by Kok and Domingos (2009). LHL receives as input the set of clause candidates considered by relational pathfinding (Richards and Mooney, 1992) and focuses only on the most promising ones. Developed in the ILP community, relational pathfinding searches for clauses by tracing paths across the true instantiations of relations in the data. The data are represented as a generalized graph, called hypergraph, having edges connecting any number of nodes, called hyperedges. However, because in real-world relational domains the search space over relational paths may be very large, a crucial aspect of LHL is that it does not perform relational pathfinding over the original relational graph of the data, but over a so-called lifted hypergraph, which is formed by jointly clustering the entities in the domain via an agglomerative clustering procedure. Intuitively, constants are clustered together if they tend to participate in the same kind of relations (i.e. predicates) with constants from other clusters. The complexity of the agglomerative clustering in the general case is $O(n^3)$, which makes it slow for large datasets. On the other hand, pathfinding on this lifted hypergraph is typically at least an order of magnitude faster than on the ground training data, producing MLNs that are more accurate than previous approaches. Subsequently, the ground atoms in each path are variabilized, and they are used to form clauses, which are evaluated using a pseudo-likelihood measure. Then, the algorithm iterates over the clauses from shortest to longest and for each clause, compares its score against those of its sub-clauses. If a clause scores higher than all the sub-clauses it is retained, otherwise it is discarded because it is unlikely to be useful. Finally, the retained clauses are added to an MLN, the weights are relearnt and the clauses are kept in the MLN, which improves the overall weighted pseudo log-likelihood. LHL is a data-driven algorithm which cannot exploit background knowledge for learning rules. This inability makes it inappropriate for capturing complex relations and learning qualitatively meaningful

rules.

Structural Motifs Kok and Domingos (2010) have proposed constraining the search for clauses by identifying so-called *structural motifs*, which capture commonly occurring patterns among densely connected entities in the domain (i.e. sets of entities that are closely related). The Learning using Structural Motifs (LSM) algorithm, a modification of LHL, proceeds by first identifying motifs (recurring patterns) and then searching for clauses by performing relational pathfinding within them. To discover motifs, LSM starts from an entity i in the relational graph and performs a series of random walks. The random walks are used to calculate the average number of steps required to reach another entity j for the first time, called hitting time. Entities that are reachable within a thresholded hitting time as well as the hyperedges among them are included in the motif and the paths through which they are reachable from i are recorded. Next, the entities included in the motif are clustered by their hitting times into groups of potentially symmetrical nodes (i.e. nodes that have symmetrical paths). The nodes within each group are then further clustered in an agglomerative manner by the similarity of distributions over paths through which they are reachable from i. This process results in a lifted hypergraph, analogous to the one produced by LHL; however, whereas in LHL nodes were clustered based on their close neighborhood in the relational graph, here they are clustered based on their longer-range connections to other nodes. Thus, intuitively, LHL is making use of length-2 paths to determine the similarity of nodes. In contrast, LSM uses longer paths, and thus more information, to find clusterings of nodes (motifs). In addition, LSM finds various clusterings rather than just a single one. Motifs are extracted from the lifted hypergraphs through depth-first search. Finally, LSM runs relational pathfinding on each motif to find candidate rules, and retains the good ones in an MLN using the same procedure as LHL. Although LSM is much faster and accurate than LHL, especially for more compex models, it, too, is data-driven and cannot exploit background knowledge for learning rules.

Gradient-Based Boosting Khot et al. (2011) have extended the functional gradient boosting approach to learning the relational dependency networks of Natarajan et al. (2012) to MLNs. In contrast to previous approaches, they learn structure and parameters simultaneously, thus avoiding the cost of repeated parameter estimation. This is done through a sequence of functional gradient steps, each of which adds clauses based on the point-wise gradients of the training examples in the current model. They present two representations for functional gradients. The first one is based on relational regression trees (Blockeel and De Raedt, 1998) and the second one learns Horn clauses by using a beam search that adds literals to clauses that reduce the squared error. Moreover, Khot et al. (2015) extend the algorithm to handle missing data by using an EM-based approach.

Markov Networks Structure learning techniques for Markov networks have been developed in the graphical models community. One technique is that of structure selection through appropriate regularization (Lee et al., 2006). In this approach, a large number of factors of a Markov network are evaluated simultaneously by training parameters over them and using the L1 norm as a regularizer (as opposed to the typically used L2 norm). Since the L1 norm imposes a strong penalty on smaller parameters, its effect is that it forces more parameters to zero, leading to sparser models. Huynh and Mooney (2008) extend this technique for structure learning of MLNs by first using Aleph (Srinivasan, 2004), an off-the-shelf ILP learner, to generate a large set of potential factors (in this case, first-order clauses), and then perform L1-regularized parameter learning over this set. Another technique proposed by Lowd and Davis (2010) uses probabilistic decision trees to learn the structure. The Decision Tree Structure Learning (DTSL) algorithm learns probabilistic decision trees using the training data in a depth-first manner to predict the value of each variable. It then converts the trees into sets of conjunctive features. All

learned features are merged into a global model and the parameters for those features can be estimated using any standard parameter learning method. In addition to this conversion, various prune methods are used during the feature generation process in order to make learning and inference faster. Lowd and Davis (2014) extended the algorithm and introduced two new variations of DTSL. The first one, DT-BLM, builds on DTSL by using the Bottom-Up Learning of Markov Networks (BLM) algorithm of Davis and Domingos (2010) to further refine the structure learned by DTSL. This algorithm is much slower but usually more accurate than DTSL. Furthermore, it serves as an example of how decision trees can be used to improve search-based structure learning algorithms, by providing a good initial structure. The second one, DT+L1, combines the structure learned by DTSL with the pairwise interactions learned by L1-regularized logistic regression (Ravikumar et al., 2010) by taking the union of the best DTSL and L1 feature set. The trees used by DTSL are good at capturing higher-order interactions. In contrast, L1 captures many independent interaction terms, but each interaction can only be between just two variables. Their combination offers the potential to represent both kinds of interaction, leading to better performance in many domains.

Online Structure Learning The Online Structure Learning (OSL) algorithm proposed by Huynh and Mooney (2011b) updates both the structure and the parameters of the model using an incremental approach based on the model's incorrect predictions. Unlike the previously presented methods, OSL takes into account the predicted possible worlds, i.e. the most probable possible worlds predicted by the current model. Specifically, at each step t of the algorithm, if the predicted possible world y_t^P is different from the ground-truth one y_t then OSL focuses on searching for clauses that differentiate y_t from y_t^P. This is related to the idea of using implicit negative examples in ILP (Zelle et al., 1995). In this particular case, each ground-truth possible world plays the role of a positive example and any predicted possible world that differs from the ground-truth one is incorrect and can be considered as a negative example. In addition, this follows the max-margin training criterion which focuses on discriminating the true label from the most probable incorrect one (Tsochantaridis et al., 2005). In order to discover useful clauses specific to the set of wrongly predicted atoms, OSL employs relational pathfinding over a hypergraph (Richards and Mooney, 1992). The search procedure is also combined with mode declarations (Muggleton, 1995), a form of language bias, to speed up the process. Paths found by the mode-guided relational pathfinding process are generalized into first-order clauses by replacing constants with variables. The resulting set of clauses is added to an MLN and the parameters are updated using the AdaGrad weight learning algorithm described in Section 3.2.

The aforementioned methods for structure learning in Markov Logic Networks, excluding OSL, are batch learning algorithms that are effectively designed for training data consisting of many ground instances. Moreover, most of these algorithms are strictly data-driven, which means that they only consider the ground-truth possible worlds and search for clauses that improve the likelihood of those worlds. They do not exploit the background knowledge that may be available about a task and may spend a lot of time exploring clauses that are true in most of these worlds, therefore largely useless for the purposes of learning. To overcome these issues we developed OSLα, presented in Section 3.4, that exploits the background knowledge domain-independent axiomatization in order constrain the seach space of possible structures. Moreover, it uses an online strategy, like OSL, in order to effectively handle large datasets.

3.4 OSLα: Online Structure Learning using background knowledge Axiomatization

Our goal is to effectively learn Event Calculus definitions, in order to accurately perform event recognition given an input of observed SDEs. OSL has several limitations which renders it unable to learn such definitions. Specifically, even when performing mode-guided search over the hypergraph, the space of possible paths can become exponentially large. For instance, the Event Calculus is a temporal formalism describing CE occurrences over time. Therefore, data used for training will inevitably contain a large domain of time points (possibly) having multiple complex temporal relations between events. Mode declarations alone cannot handle this large domain. It will be then fundamental to prune a portion of the search space. Moreover, existing structure learning methods, including OSL, assume that domains do not contain any functions, which are essential for the Event Calculus representation.

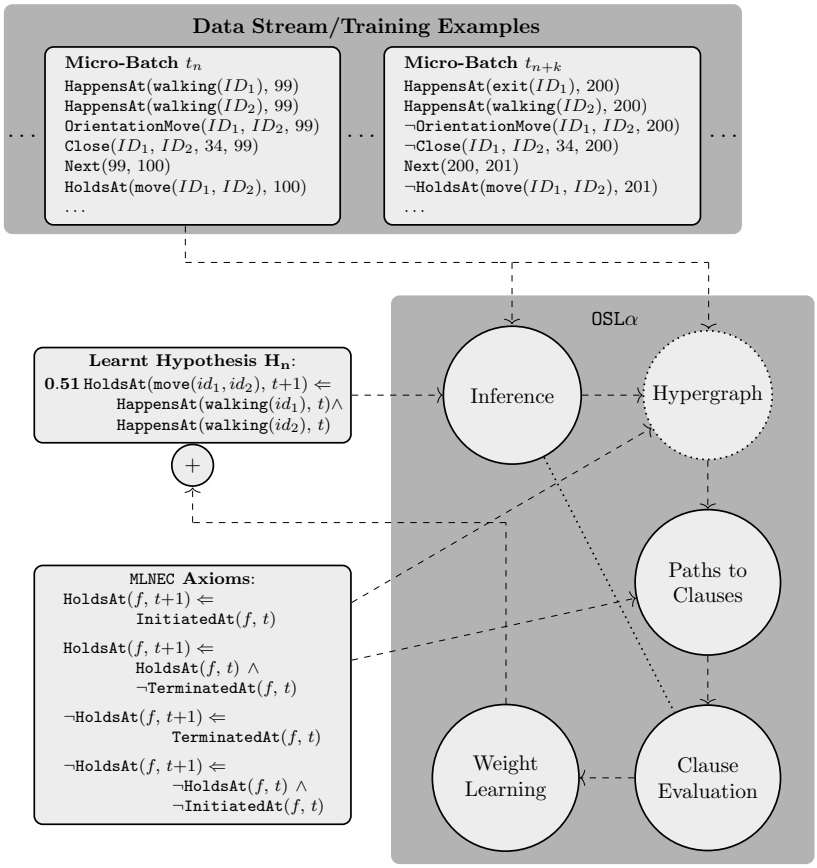

Figure 3.1: The procedure of OSLα

To cope with these limitations we propose OSLα, which exploits domain-independent axioms of the background knowledge, to further constrain the search space only to clauses subject to specific characteristics introduced by these axioms. Furthermore, our approach can handle a subset of functions defined by the first-order logic formalism and effectively learn definitions using these functions. Figure 3.1 presents the interactions among the components underlying OSLα. The background knowledge consists of the MLN–EC axioms (i.e. domain-independent rules) and an already known hypothesis (i.e. set of clauses). At a step t_n of the online procedure a training example (micro-batch) $\mathcal{D}_{t_n}$ arrives and is used together with the already learnt hypothesis in order to predict the truth values $y^P_{t_n}$ of the CEs of

interest using probabilistic inference. Then for all wrongly predicted CEs the hypergraph is searched, guided by MLN–EC axioms, for definite clauses explaining these CEs. The paths discovered during the search are then translated into clauses and evaluated. The resulting set of retained clauses is passed onto the weight learning module for estimating their weights. Then, the set of weighted clauses is appended into the hypothesis $\boldsymbol{\mathcal{H}}_n$ and the whole procedure is repeated given the next training example $\boldsymbol{\mathcal{D}}_{t_{n+1}}$.

In Section 3.4.1 we describe the procedure of grouping the axioms into templates used by the hypergraph to constrain the space of possible structures. In Section 3.4.2 we present the hypergraph search by relational pathfinding and mode declarations alone and in Section 3.4.3 the extended template-guided search using the MLN–EC axioms. Finally, in Section 3.4.4 the clause creation/evaluation and weight learning procedure is presented.

3.4.1 Extract Templates from Axioms

OSLα begins by partitioning the background knowledge into a set of axioms $\mathcal{A}$ and a set $\mathcal{B}$ incorporating the rest of the formulas. Each axiom $\alpha \in \mathcal{A}$ should contain exactly one so-called *template predicate* as well as at least one *query predicate* representing the CEs we are trying to recognize. These are the desired properties required for OSLα in order to operate. Furthermore, axioms must not contain free variables, meaning variables only appearing in a single predicate. In order to explain these properties in detail, consider that $\mathcal{A}$ contains the four axioms of Event Calculus (described in D3.1) presented below,

$$\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{InitiatedAt}(f, t) \tag{3.5}$$

$$\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{HoldsAt}(f, t) \wedge \neg\texttt{TerminatedAt}(f, t) \tag{3.6}$$

$$\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{TerminatedAt}(f, t) \tag{3.7}$$

$$\neg\texttt{HoldsAt}(f, t{+}1) \Leftarrow \neg\texttt{HoldsAt}(f, t) \wedge \neg\texttt{InitiatedAt}(f, t) \tag{3.8}$$

As stated previously in Skarlatidis et al. (2014), axioms (3.5) and (3.6) determine when a fluent F holds. Similarly, (3.7) and (3.8) determine when F does not hold. Then, $\texttt{HoldsAt} \in \mathcal{Q}$ are the *query predicates* and $\texttt{InitiatedAt}, \texttt{TerminatedAt} \in \mathcal{P}$ are the *template predicates*. Those latter predicates specify the conditions under which a CE starts and stops being recognized respectively. They form the target CE patterns that we want to learn. Therefore, these four axioms of MLN–EC can be used to define a template over all possible structures and guide the search in selecting clauses following the desired properties of the Event Calculus. By exploiting the information of this template, the algorithm does not need to search over time sequences during relational pathfinding, but only needs to find explanations for the template predicates over the current time-point, in the form of definite clauses. Following the work of Skarlatidis et al. (2015a), we perform circumscription by *predicate completion*, a syntactic transformation where formulas are translated into stronger ones. Predicate completion is applied on InitiatedAt and TerminatedAt predicates. Finally, we also eliminate the InitiatedAt and TerminatedAt predicates by exploiting the equivalences resulting from predicate completion (Mueller, 2008). Formally speaking, the algorithm only needs to search for definite clauses having the following form:

$$\texttt{InitiatedAt}(f, t) \Leftarrow \text{body}$$
$$\texttt{TerminatedAt}(f, t) \Leftarrow \text{body}$$

The body of these definitions is a conjunction of n literals $\ell_1 \wedge \cdots \wedge \ell_n$, which is very convenient because it can be seen as a variabilized hypergraph path as we shall explain below. Given a set of axioms $\mathcal{A}$, we further partition it into templates. Each template $\mathcal{T}_i$ contains axioms having identical Cartesian product of domain types over their template predicate variables. For instance, MLN–EC axioms (3.5) – (3.8) should all belong to one template because `InitiatedAt` and `TerminatedAt` both have joint domain `Fluent` $\times$ `Time`. Each of these resulting templates $\mathcal{T}_i$ is used during relational pathfinding in order to constrain the search space into specific bodies for the definite clauses.

3.4.2 Hypergraph and Relational Pathfinding

Following the procedure of OSL, at each step t the algorithm receives an example $\mathbf{x}_t$ representing the evidence, it produces the predicted label $\mathbf{y}_t^P = \text{argmax}_{\mathbf{y}\in\boldsymbol{\mathcal{Y}}}\langle \mathbf{w}, \mathbf{n}(\mathbf{x}_t, \mathbf{y})\rangle$, and then receives the true label $\mathbf{y}_t$. Given both $\mathbf{y}_t$ and $\mathbf{y}_t^P$, in order to discover clauses that separate $\mathbf{y}_t$ from $\mathbf{y}_t^P$, it finds all ground atoms that are in $\mathbf{y}_t$ but not in $\mathbf{y}_t^P$ denoted as $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$. That means Δy_t contains the false positives and false negatives resulted from inference. Then, it searches the ground-truth possible world ($\mathbf{x}_t$,$\mathbf{y}_t$), namely the training example of the current step t, for clauses specific to the axioms defined in the background knowledge using the constructed templates. In contrast to OSL, OSLα considers all misclassified (false positives/negatives) ground atoms instead of the true ones (false negatives), and searches the ground-truth possible world for clauses.

In order to discover useful clauses specific to a set of wrongly predicted atoms, we employ relational pathfinding, which considers a training example $\mathcal{D}$ as a hypergraph having constants as nodes and true ground atoms as hyperedges, connecting the nodes appearing as its arguments. Hyperedges are a generalization of edges connecting any number of nodes. It then searches the hypergraph for paths that connect the arguments of an input literal. Consider, for example, a training example $\mathcal{D}_t$ at step t of SDEs and CEs:

SDEs :
`Minimum_Activity(Min_Activity_Lid1, Lid1, Middle)`
`Inactive(Inactive_Lid1, Lid1, Ramp)`
`Happens(Inactive_Lid1, 2)`
`Happens(Min_Activity_Lid1, 2)`
`Next(2, 3)`
CEs :
`HoldsAt(Congested, 3)`
`HoldsAt(Idle, 3)`

The equivalent hypergraph representing the above training example is presented in Figure 3.2, where each colored ellipse is a hyperedge and each dot is a constant node. Starting from each wrongly predicted ground atom in Δy_t, relational pathfinding searches for all paths (up to length l) connecting the

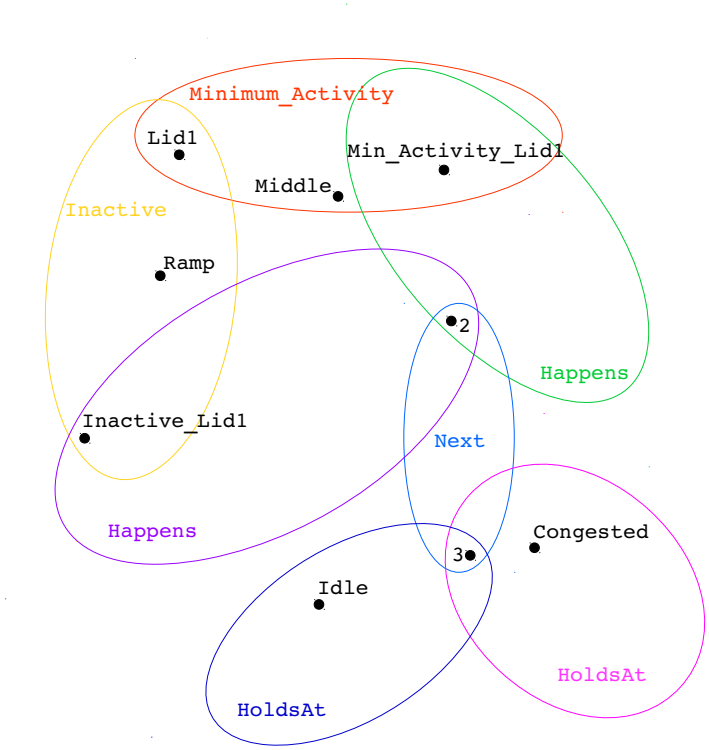

Figure 3.2: Simple hypergraph

arguments of the given atom. In our case, these ground atoms correspond to the wrongly predicted CEs. A path of hyperedges corresponds to a conjunction of true ground atoms connected by their arguments and can be generalized into conjunction of variabilized literals. For example consider the training example presented above. If the predicted label y_t^P says that $\texttt{HoldsAt}(\texttt{Idle}, 3)$ is false, then is considered a wrongly predicted atom and therefore the hypergraph should be searched for paths. Below, we present two paths found by searching the hypergraph of Figure 3.2 for paths up to length $l = 4$ for this misclassified CE. These paths are also presented in Figure 3.3 with highlighted hyperedges. Obviously there are many other possible paths that can be found.

$$\{\texttt{HoldsAt}(\texttt{Idle}, 3), \texttt{Next}(2, 3), \texttt{Happens}(\texttt{Inactive\_Lid1}, 2), \tag{3.9}$$
$$\texttt{Inactive}(\texttt{Inactive\_Lid1}, \texttt{Id1}, \texttt{Ramp})\}$$

(3.10)

$$\{\texttt{HoldsAt}(\texttt{Congested}, 3), \texttt{Next}(2, 3), \texttt{Happens}(\texttt{Min\_Activity\_Lid1}, 2), \tag{3.11}$$
$$\texttt{Minimum\_Activity}(\texttt{Min\_Activity\_Lid1}, \texttt{Lid1}, \texttt{Middle})\}$$

To speed up relational pathfinding, we employ mode declarations to constrain the search for paths, which represent the body of the definite clauses defined by the template predicates appearing in the axioms of the background knowledge. Mode declarations are a form of language bias that constrain the search for definite clauses. Since we want to constrain the space of paths, we use a variant of the path mode declarations introduced by Huynh and Mooney (2011b). Formally speaking, declaration $\texttt{modep}(r, p)$ has two components: a recall number $r \in \mathbb{N}_0$, and an atom p whose arguments are place-markers optionally preceding the symbol $\#$. A place-marker is either $+$ (input), $-$ (output), or . (ignore).

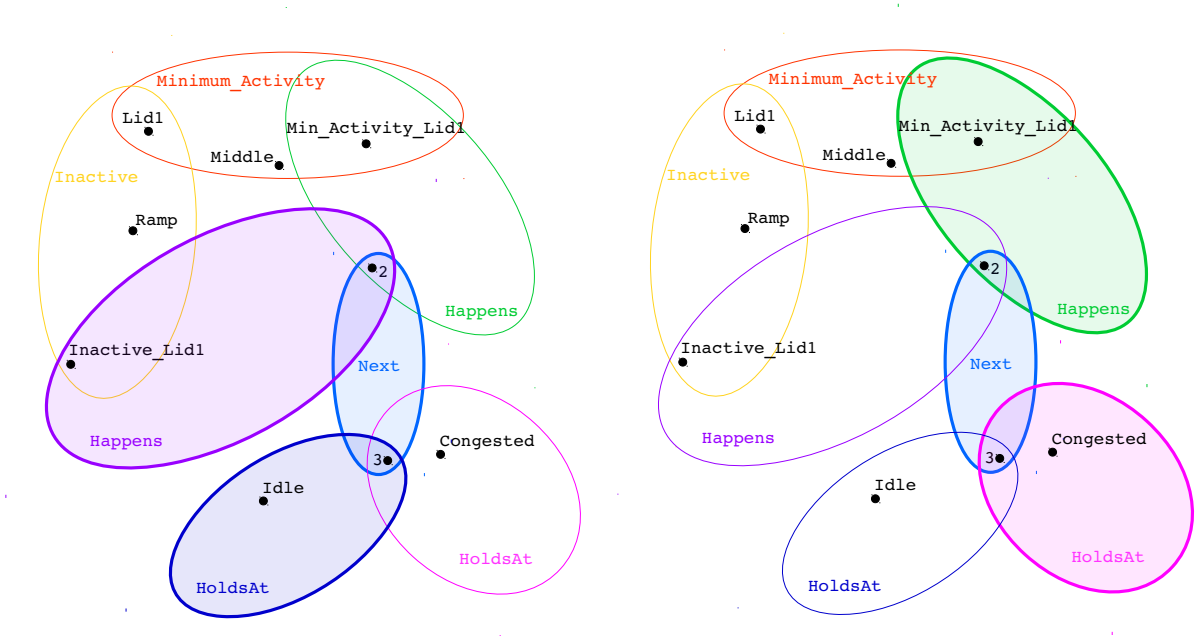

Figure 3.3: Hyper-graph paths discovered by relational pathfinding

The symbol # preceding each place-marker specifies that this particular predicate argument will remain constant after the generalization of the path. For many tasks it is critical to have clauses specific to particular constants. The recall number r limits the number of appearances of the predicate p in a path to r. These place-markers restrict the search of relational pathfinding. A ground atom is only added to a path if one of its arguments has previously appeared as 'input' or 'output' arguments in the path and all of its 'input' arguments are 'output' arguments of previous atoms.

Furthermore, in order for our algorithm to be able to handle functions, we also introduce mode declarations for functions, defined as $\texttt{modef}(r, p)$ and having exactly the same functionality as described above. The only difference is that functions have a return type and the position for this type cannot be declared in the mode declaration; it is always assumed as 'input'. The intuition behind this is that a function always returns a constant value belonging to some other function or predicate as an argument. Therefore, the return value it must appear before in the path. Below we present a couple of mode declarations for both predicates and functions:

$$\texttt{modep}(1, \texttt{HappensAt}(-, +)) \quad \texttt{modef}(1, \texttt{inactive}(-, -))$$

The above mode declarations require that a legal path contains at most one ground atom of each of the predicates `HappensAt` and at most one ground function `inactive`. Moreover, the first argument of `HappensAt` and all arguments of `inactive` are 'output' arguments; the second argument of `HappensAt` is an 'input' argument. The 'input' mode constrains the position constant in `HappensAt` atoms to have appeared in previous atoms in a path.

Algorithm 1 presents the pseudocode for efficiently constructing the hypergraph from a training example $\mathcal{D}$ based on mode declarations, by only allowing input and output nodes. There is no point in constructing the entire search space, because only the portion of it defined by the mode declarations will be eventually searched. Note that template predicates are not added in the hypergraph because they are not allowed to appear in the body of the definite clause. Moreover, in order to search for functions, before the hypergraph is constructed, all functions in the domain are converted into auxiliary

Algorithm 1 HG($\mathcal{D}$, modes, $\mathcal{P}$)

Input: $\mathcal{D}$: training example, modes: mode declarations, $\mathcal{P}$: set of *template predicates*
Output: HG: hypergraph

```
for each constant c ∈ D do
    HG[c] = ∅
for each true ground atom p(c_1, ..., c_r) ∈ D & p ∉ P do
    for each constant c_i ∈ {c_1, ..., c_r} do
        if isInputOrOutputVar(c_i, modes) then
            HG[c_i] = HG[c_i] ∪ {p(c_1, ..., c_r)}
return HG
```

predicates and their corresponding mode declarations are converted into predicate modes. For example, the function `inactive` will be converted into the auxiliary predicate `Inactive` (see training example above) having arity 3, because it also incorporates the return value of the function and the corresponding mode declaration $\texttt{modef}(1, \texttt{inactive}(-,-))$ will be converted into $\texttt{modep}(1, \texttt{Inactive}(+,-,-))$. By performing this conversion, the functions can be included in the hypergraph as auxiliary predicates and added to paths during the search procedure.

3.4.3 Template Guided Search

Starting from each wrongly predicted ground atom $q(c_1, \ldots, c_n) \in \Delta y_t$, we use the templates $\mathcal{T}_i$ constructed at the initial steps of the algorithm in order to find the corresponding ground template predicates for which the axioms belonging in the template $\mathcal{T}_i$ are satisfied by the current training example. The algorithm considers each axiom $\alpha \in \mathcal{A}$ in turn and checks whether the desired properties, presented in Section 3.4.1, hold. Assume, for example, that one of these axioms is 3.5:

$$\texttt{HoldsAt}(f, t{+}1) \Leftarrow \texttt{Next}(t, t+1) \wedge \texttt{InitiatedAt}(f, t)$$

and we have given the wrongly predicted ground atom $\texttt{HoldsAt}(CE, T_4)$ (false negative). We would substitute the constants of the given atom q into the axiom. The result of the substitution on the above rule will be the following partially grounded axiom:

$$\texttt{HoldsAt}(\texttt{CE}, \texttt{T}_4) \Leftarrow \texttt{Next}(t, \texttt{T}_4) \wedge \texttt{InitiatedAt}(\texttt{CE}, t)$$

If after the substitution there are no variables left in the template predicate of the axiom, it adds the grounded template predicate into the initiation set $\mathcal{I}$ and moves to the next axiom. Otherwise, it searches for all literals in the axiom sharing variables with the template predicate. For these literals, it searches the training data for all jointly ground instantiations among those satisfying the axiom. Then, for each of them, it substitutes the remaining variables of the template predicate and adds them into the initiation set. In this example, `InitiatedAt` has one remaining variable t and the only literal sharing this variable is `Next`. Thus, we search for all ground instantiations of `Next` in the training data $\mathcal{D}$ that satisfy the axiom and substitute their constants into the partially grounded axiom. Because t represents time-points and the predicate `Next` describes sequence of time-points, there will be only one true ground

atom in the training data $\mathcal{D}$ having the constant T_3. The same applies for axioms 3.7 and 3.8 determining the termination conditions and false positives. Algorithm 2 presents the pseudocode for extracting these ground template predicates.

Algorithm 2 InitialSet(q, $\mathcal{D}$, $\mathcal{T}$)

Input: q: misclassified ground atom, $\mathcal{D}$: training example, $\mathcal{T}$: path template
Output: $\mathcal{I}$: a set of grounded template predicates

$\mathcal{I} = \emptyset$ ▷ The set of ground template predicates used to initiate the search
for each axiom $\alpha \in \mathcal{T}$ **do**
  if $\exists$ literal $\ell \in \alpha$: signature(ℓ) = signature(q) & isPositive(ℓ) == isTrue($g, \mathcal{D}$) **then**
    θ-substitute $\beta = \alpha\theta$ where $\theta = \{\text{variables}(\ell) \rightarrow \text{constants}(q)\}$
    $\tau = \text{templateAtom}(\beta)$
    if $\exists$ variable $v \in \tau$ **then**
      $\mathcal{L} = \emptyset$
      if $\exists$ variable $v \in \tau \wedge \exists$ literal $\ell \in \beta : v \in \ell$ **then**
        $\mathcal{L} = \mathcal{L} \cup \ell$
      for each literal $\ell \in \mathcal{L}$ **do**
        $\mathbf{T}_l = \{c_1, \ldots, c_n\} \in \mathcal{D} : \ell(c_1, \ldots, c_n) \Rightarrow \top$
      for each row $r = \{c_1, \ldots, c_n\} \in \bowtie_{\ell=1}^{|\mathcal{L}|} \mathbf{T}_l$ **do**
        θ-substitute $\gamma = \beta\theta$ where $\theta = \{variables(\beta) \Rightarrow r\}$
        $\mathcal{I} = \mathcal{I} \cup \text{templateAtom}(\gamma)$
    else
      $\mathcal{I} = \mathcal{I} \cup \tau$
return $\mathcal{I}$

For each grounded template predicate returned by Algorithm 2, the mode-guided relational pathfinding, presented in Algorithm 3, is used to search the constructed hypergraph for an appropriate body. It recursively adds to the path ground atoms or hyperedges that satisfy the mode declarations. The search terminates when the path reaches a specified maximum length or when no new hyperedges can be added. The algorithm stores all paths encountered during the search.

By employing this procedure the hypergraph is essentially pruned in order to contain only ground atoms explaining the template predicates. Consider the hypergraph presented in Figure 3.2. By exploiting the Event Calculus axioms the hypergraph is pruned and only need to contain predicates explaining the `InitiatedAt` and `TerminatedAt` as presented in Figure 3.4. Therefore the paths (3.9) and (3.11) are pruned by removing `Next` and `HoldsAt` predicates, resulting into the paths (3.12) and (3.13). The pruning resulting from the template guided search is essential to learn Event Calculus, because the search space becomes independent of time.

$$\{\texttt{InitiatedAt(Idle,3)}, \texttt{Happens(Inactive\_Lid1,2)}, \texttt{Inactive(Inactive\_Lid1,Id1,Ramp)}\} \tag{3.12}$$

$$\{\texttt{InitiatedAt(Congested,3)}, \texttt{Happens(Min\_Activity\_Lid1,2)}, \texttt{Minimum\_Activity(Min\_Activity\_Lid1,Lid1,Middle)}\} \tag{3.13}$$

Algorithm 3 ModeGuidedSearch(curPath, $\mathcal{C}$, HG, mode, maxLength, paths)

Output: paths: a set of paths

if |curPath| $<$ maxLength **then**
  for each constant $c \in \mathcal{C}$ **do**
    for each $\mathrm{p}(c_1, \ldots, c_r) \in HG[c]$ **do**
      if canAdd(p, curPath, mode) **then**
        if curPath $\notin$ paths **then**
          curPath $=$ curPath $\cup\{\mathrm{p}(c_1, \ldots, c_r)\}$
          paths $=$ paths $\cup\{p(c_1, \ldots, c_r)\}$
          $\mathcal{C}' = \emptyset$
          for each $c_i \in \{c_1, \ldots, c_r\}$ **do**
            if $c_i \notin \mathcal{C}$ & isInputOrOutputVar(c_i, mode) **then**
              $\mathcal{C} = \mathcal{C} \cup \{c_i\}$
              $\mathcal{C}' = \mathcal{C}' \cup \{c_i\}$
          ModeGuidedFindPath(curPath,$\mathcal{C}$,HG,mode,maxLength,paths)
          curPath $=$ curPath $\setminus$p
          $\mathcal{C} = \mathcal{C} \setminus \mathcal{C}'$

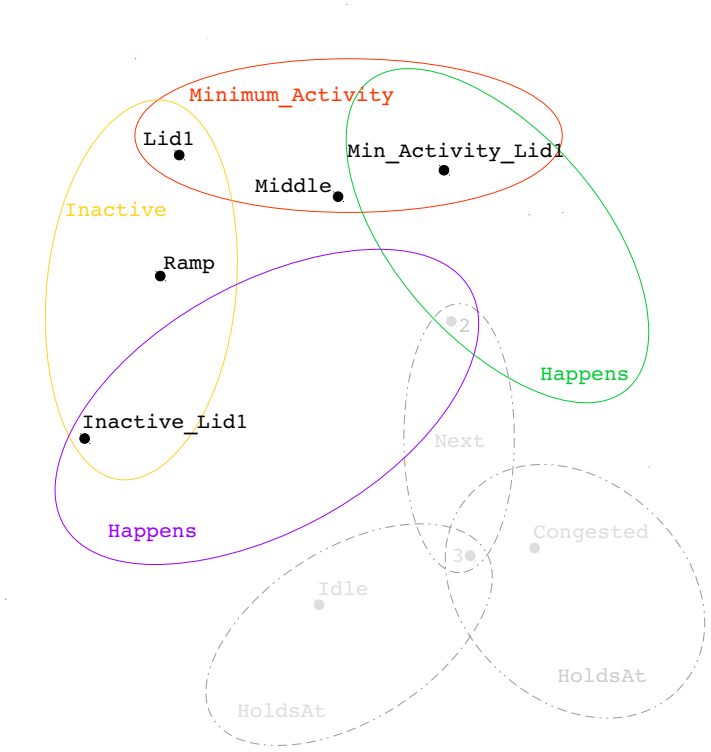

Figure 3.4: Pruned hypergraph

3.4.4 Clause Creation, Evaluation and Weight Learning

In order to generalize paths into first-order clauses, we follow three steps. First, we replace each constant c_i in a conjunction with a variable v_i, except for those declared constant in the mode declarations. Then, these conjunctions are used as a body to form definite clauses using the template predicate present in each path as head. The auxiliary predicates are converted back into functions. Therefore, from the paths presented above, the following definite clauses will be created:

$$\texttt{InitiatedAt}(\texttt{Idle}, t) \Leftarrow \texttt{HappensAt}(\texttt{inactive}(lid_1), t) \tag{3.14}$$

$$\texttt{InitiatedAt}(\texttt{Congestion}, t) \Leftarrow \texttt{HappensAt}(\texttt{minimum\_activity}(lid_1), t) \tag{3.15}$$

The first definition says that the CE `Idle` is initiated for a lane lid_1 at t when the event `inactive` happens for lid_1. Similarly the second definition says that `Congestion` is initiated when lid_1 has minimum activity. These definite clauses can be used together with the axioms defined in the background knowledge in order to eliminate all the template predicates by exploiting the equivalences resulting from predicate completion. After the elimination process all resulting formulas are converted into clausal normal form (CNF), since that is the form used by the inference algorithms. Therefore the resulting set of clauses is independent of the template predicates. Evaluation must take place for each clause c individually. The difference in the number of true groundings of c in the ground-truth world $(\mathbf{x}_t, \mathbf{y}_t)$ and the predicted world $(\mathbf{x}_t, \mathbf{y}_t^P)$ is computed. Then, the only clauses whose difference in the number of groundings is greater than or equal to a predefined threshold μ will be added to the existing MLN:

$$\Delta n_c = n_c(\mathbf{x}_t, \mathbf{y}_t) - n_c(\mathbf{x}_t, \mathbf{y}_t^P)$$

The intuition behind this measure is to keep clauses whose coverage of the ground-truth world is different from the one induced by the clauses already learnt.

Finally, the parameters of these clauses are learned by the AdaGrad online learner described in Section 3.2. We can treat the rules as either hard or soft constraints and modify the behavior of the Event Calculus as stated by Skarlatidis et al. (2015a); Skarlatidis (2014). In order to do this, we only need to define the corresponding axioms of the MLN–EC as either soft or hard constraints in the background knowledge. By doing so, the resulting set of clauses produced by predicate completion and elimination can be either soft or hard constraint, depending on the axiom used to produce each clause during predicate completion. If the axiom has an infinite weight, then the corresponding clauses created by it will also have an infinite weight and therefore it cannot be changed during weight learning.

Finally, at the end of the OSLα learning we can choose to remove those clauses whose weights are less than a predefined threshold θ in order to speed-up inference. By doing so and finding a good θ, the resulting hypothesis can be pruned significantly and the accuracy, although reduced, remains in acceptable levels. We present below the complete procedure of OSLα.

Algorithm 4 OSLa($\mathcal{C}$,$\mathcal{P}$,modes,maxLength,μ,λ,η,δ)

Input: $\mathcal{KB}$: initial knowledge base
1: $\mathcal{P}$: template predicates
2: modes: mode declarations
3: maxLength: maximum number of hyperedges in a path
4: μ: evaluation threshold
5: λ, η, δ: AdaGrad parameters
6: Partition $\mathcal{KB}$ into $\mathcal{A}$ and $\mathcal{B}$
7: Create templates $\boldsymbol{\mathcal{T}}$ from $\mathcal{A}$
8: Initialize resulting theory $\mathcal{R}_0 = \mathcal{B}$
9: Initialize weight vector $\mathbf{w}_0 = \text{initialValue}$
10: **for** $t = 1$ to T **do**
11: Receive an instance $\mathbf{x}_t$
12: Predict $\mathbf{y}_t^P = \text{argmax}_{\mathbf{y}\in\boldsymbol{\mathcal{Y}}}\langle \mathbf{w}, \mathbf{n}(\mathbf{x}_t, \mathbf{y})\rangle$
13: Receive the correct target $\mathbf{y}_t$
14: Compute $\Delta y_t = \mathbf{y}_t \setminus \mathbf{y}_t^P$
15: **if** $\Delta y_t \neq \emptyset$ **then**
16: $HG = \text{HG}\big((\mathbf{x}_t, \mathbf{y}_t), \text{modes}, \mathcal{P}\big)$
17: paths= $\emptyset$
18: **for each** wrogly predicted query atom $\text{q} \in \Delta y_t$ **do**
19: **for each** template $\mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ **do**
20: $\mathcal{I} = \text{InitialSet}\big(\text{q}, (\mathbf{x}_t, \mathbf{y}_t), \mathcal{T}_i\big)$
21: $\mathcal{V} = \emptyset$
22: **for each** $\tau(c_1, \ldots, c_n) \in \mathcal{I}$ **do**
23: **for each** $c_i \in \{c_1, \ldots, c_n\}$ **do**
24: **if** $\text{isInputOrOutputVar}(c_i, \text{modes})$ **then**
25: $\mathcal{V} = \mathcal{V} \cup c_i$
26: $\text{ModeGuidedSearch}\big(\text{q}, \mathcal{V}, HG, \text{modes}, \text{maxLength}, \text{paths}\big)$
27: $\mathcal{D}_t = \text{CreateDefinitions}(\mathcal{D}_{t-1}, \text{paths}, \text{modes})$
28: $\mathcal{C}_t = \text{CreateClauses}(\mathcal{D}_t, \text{modes})$
29: Compute $\Delta\mathbf{n}_{\mathcal{C}_t}$
30: **for** $i = 1$ to $|\mathcal{C}_t|$ **do**
31: **if** $\Delta\mathbf{n}_{\mathcal{C}_{t,i}} \leq \mu$ **then**
32: Remove definite clause d_i corresponding to $\mathcal{C}_{t,i}$ from D_t
33: $\mathcal{R}_t = \mathcal{B} \cup \text{CreateClauses}(\mathcal{D}_t, \text{modes})$
34: Compute $\Delta\mathbf{n}_{\mathcal{R}_t}$
35: **for** $i = 1$ to $|\mathcal{R}_t|$ **do**
36: **if** $\exists\, c_{i,t-1} \in \mathcal{R}_{t-1},\ c_{i,t} \in \mathcal{R}_t : c_{i,t-1}\ \theta$-subsumes $c_{i,t}$ **then**
37: $w_{i,t}$=$w_{i,t-1}$
38: **else**
39: $w_{i,t}$=initialValue
40: $\text{AdaGrad}(\mathbf{w}_t, \Delta\mathbf{n}_{\mathcal{R}_t}, \lambda, \eta, \delta)$

4

Machine Learning: Experimental Evaluation

In this section we evaluate our employed parameter learning methods in the domain of video activity recognition and traffic management. Furthermore, we present results of our recently developed OSLα method for probabilistc structure learning on the surrogate dataset of video activity recognition. The aim of the experiments is to assess the effectiveness of our methods in learning both the structure and their parameters used for recognising CEs, based on imperfect CE definitions and in the presence of incomplete narratives of SDEs. To demonstrate the methods that we developed in this work, we use the publicly available benchmark dataset of the CAVIAR project[1] as well as real and simulation data provided by CNRS.

4.1 Traffic Management (Real Rocade Data)

In this task, the aim is to recognize traffic congestions which take place in the Grenoble South Ring, that links the city of Grenoble from the south-west to the north-east, by exploiting real time data collected from traffic sensors. The dataset comprises one month of data ($\approx$ 3.3GiB of sensor readings), where each day is annotated by human traffic controllers for traffic congestions. The real data was collected from sensors placed in 19 collection points along a 12km stretch of the highway and each collection point has a sensor per lane. Sensor data are collected every 15 seconds, containing the total number of vehicles passing through a lane, the average speed and sensor occupancy. These readings constitute the simple derived events (SDEs) that concern activity on individual points in the highway.

4.1.1 Learning Challenges

At first, we attempted to perform parameter learning on simple rules defining the concept of traffic congestion for any possible location regardless of the lane type. These rules are presented below:

[1] http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1

$$\texttt{InitiatedAt}(\texttt{trafficjam}(locID),\ t) \Leftarrow \\ \texttt{HappensAt}(\texttt{aggr}(locID, occupancy, avgspd),\ t) \wedge \\ 20 \leq occupancy \leq 50 \wedge 0 \leq avgspd < 50 \tag{4.1}$$

$$\texttt{InitiatedAt}(\texttt{trafficjam}(locID),\ t) \Leftarrow \\ \texttt{HappensAt}(\texttt{aggr}(locID, occupancy, avgspd),\ t) \wedge \\ 30 \leq occupancy \leq 75 \wedge 0 \leq avgspd < 35 \tag{4.2}$$

We defined a rule (4.1) recognizing that when the average speed is between 0 and 50 km/h and the sensor occupancy is between 20% and 50% in any lane of a specific location then a traffic jam is initiated in that location, as well as another rule (4.2) defining exactly the same thing but using stricter thresholds for average speed and sensor occupancy. The intuition behind these definitions is that if any location and lane have high occupancy and low average speed then probably a traffic jam takes place.

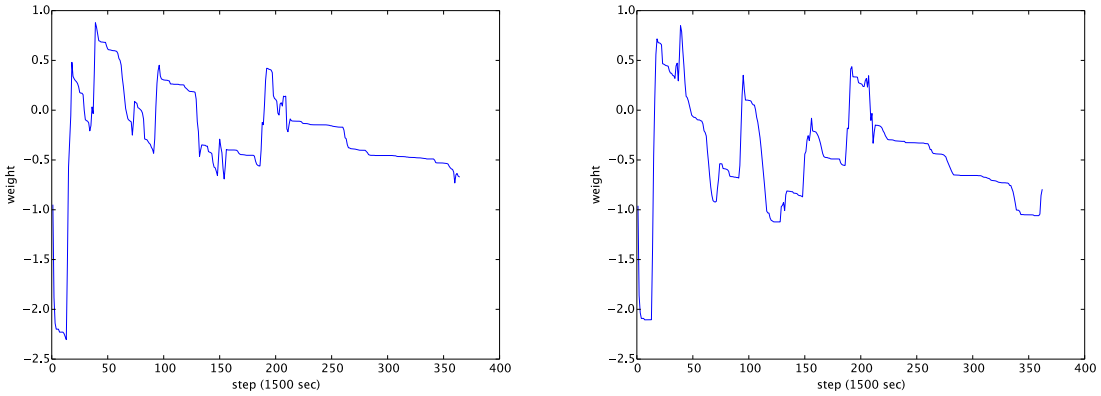

Figure 4.1: Weight value per learning step for rule (4.1) (left) and (4.2) (right)

We trained both rules for 365 learning steps, each step corresponding to 1500 seconds. We observed that the parameter value for each of the rules between sequential steps had large fluctuations leading to zero crossings, which indicate that both rules are insufficient. Zero crossings are shown in Figure 4.1. These zero crossings mean that the rules correctly capture the concept of traffic jam for some locations and completely fail for others.

In order to understand the problem better we performed a data analysis on the dataset. We plotted the annotation across the values of average speed and sensor occupancy for each location and lane. From these figures we observed that we are unable to learn a general location-independent, lane-independent congestion rule. Figure 4.2 presents an example of average speed and occupancy difference between two lanes (fast and queue) in the same location. Note that the average speed and sensor occupancy are inversely proportional. When the average speed is decreasing the occupancy is increasing. The fast lane has useful data to detect a traffic congestion in contrast to the queue lane which is completely uninformative. This is reasonable because the definition of traffic congestion differs depending on the shape of the road in a specific point, whether the specific location is a highway interchange or being close to entry points of crowded places. Therefore a fast lane has different characteristics in a traffic congestion from another lane type and should be handled differently.

Moreover, there are also many missing annotations making the machine learning task more difficult. As shown in Figure 4.3 there are cases where annotation is missing. There are many such cases in the data leading to false recognitions and false penalization of otherwise good rules.

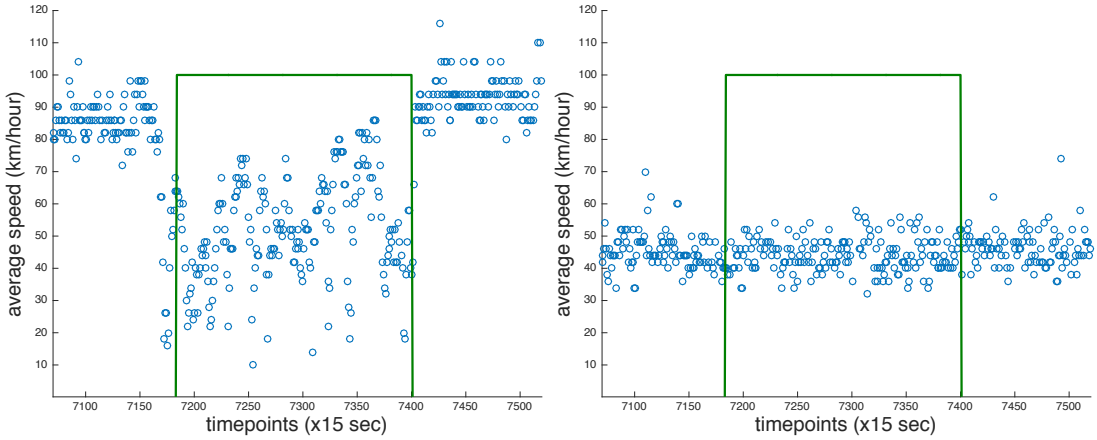

Figure 4.2: Location 0*ddd*: fast lane (left) vs queue lane (right). Blue points indicate avg. speed while green windows indicate the congestion annotated by human experts.

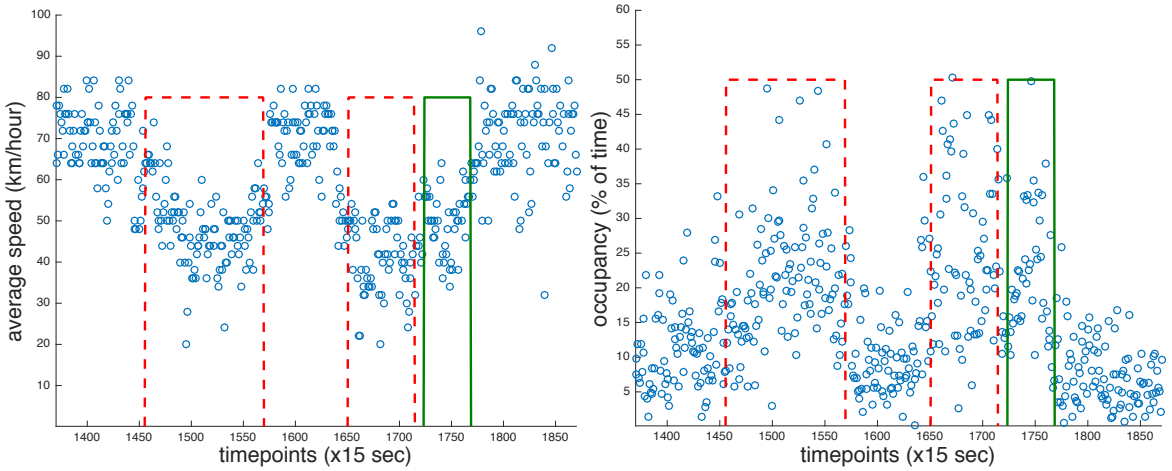

Figure 4.3: Location 25*ec*, fast lane: avg. speed (left) and occupancy (right). Blue points indicate avg. speed (occupancy), green windows the congestion annotated by human experts, and red (dashed) windows the potentially missing annotations.

4.1.2 Experimental Setup

We defined rules for traffic congestion initiation and termination for three different locations. Each pair of rules concerns a fast lane. For all these rules, we performed parameter learning for 126 steps, each consisting of 1500 seconds, that is to say about 52 hours. A pair of these initiation and termination rules are presented in equation (4.3) and (4.4) respectively.

$$\begin{aligned}&\texttt{InitiatedAt}(\texttt{trafficjam}(10314363961353708),\ t) \Leftarrow \\ &\qquad \texttt{HappensAt}(\texttt{aggr}(locID, lane, occupancy, avgspd),\ t) \wedge \\ &\qquad 26 \leq occupancy \leq 100 \wedge 0 \leq avgspd < 55 \wedge \\ &\qquad locID = 10314363961353708 \wedge lane = fast\end{aligned} \tag{4.3}$$

$$\begin{aligned}&\texttt{TerminatedAt}(\texttt{trafficjam}(10314363961353708),\ t) \Leftarrow \\ &\qquad \texttt{HappensAt}(\texttt{aggr}(locID, occupancy, avgspd),\ t) \wedge \\ &\qquad 0 \leq occupancy \leq 25 \wedge 56 \leq avgspd < 100 \wedge \\ &\qquad locID = 10314363961353708 \wedge lane = fast\end{aligned} \tag{4.4}$$

4.1.3 Experimental Results

We collected the weight values for the presented rules across all learning steps and plotted them in order to present the quality of the rules compared to the initial defined rules (4.1) and (4.2).

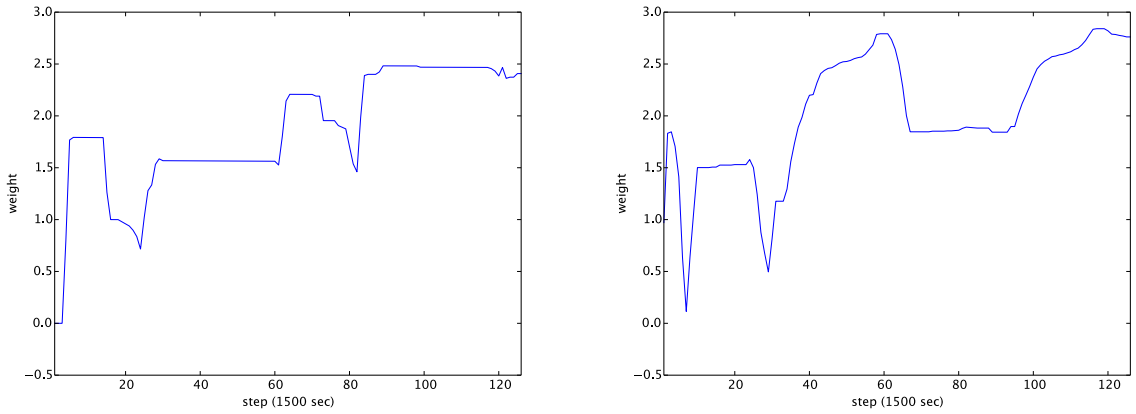

Figure 4.4: Weight value per learning step for rules (4.3) (left) and (4.4) (right)

As shown in the Figure 4.4 the fluctuations have decreased significantly, especially for the left plot representing the `InitiatedAt`. The remaining fluctuations are explained by the fact that data has incomplete annotation, which makes the learner think that in some cases it wrongly recognizes from the data traffic congestions while the annotation says otherwise. Although some fluctuations exist, there are not oscillating from positive to negative weights. This is a proof of stability for these rules, meaning that they recognizing traffic congestion most of the times.

Subsequently, we performed 10-fold cross validation over the entire dataset (172799 timepoints) using varying batch sizes. At each fold, an interval of 17280 timepoints was left out and used for testing. Figure 4.5 presents the evaluation results for OSLα and AdaGrad. In OSLα the predictive accuracy of the learned model increases initially, due to the increase in the number of learning iterations, and then decreases, due to the decreasing batch size. On the contrary, the accuracy of AdaGrad increases (almost) monotonically as the number of learning iterations increase.

OSLα achieves comparable predictive accuracy to the weighted manually constructed rules (AdaGrad), which is encouraging. Moreover, it can process data batches efficiently. For example, OLSa takes $\approx$ 11 seconds to process a 50 minute batch (1400 SDEs). As expected, AdaGrad is faster than OSLa. The predictive accuracy of the learned model, both for OSLa and AdaGrad, is low. This arises from the semi-supervised nature of the problem (see Section 4.1.1).

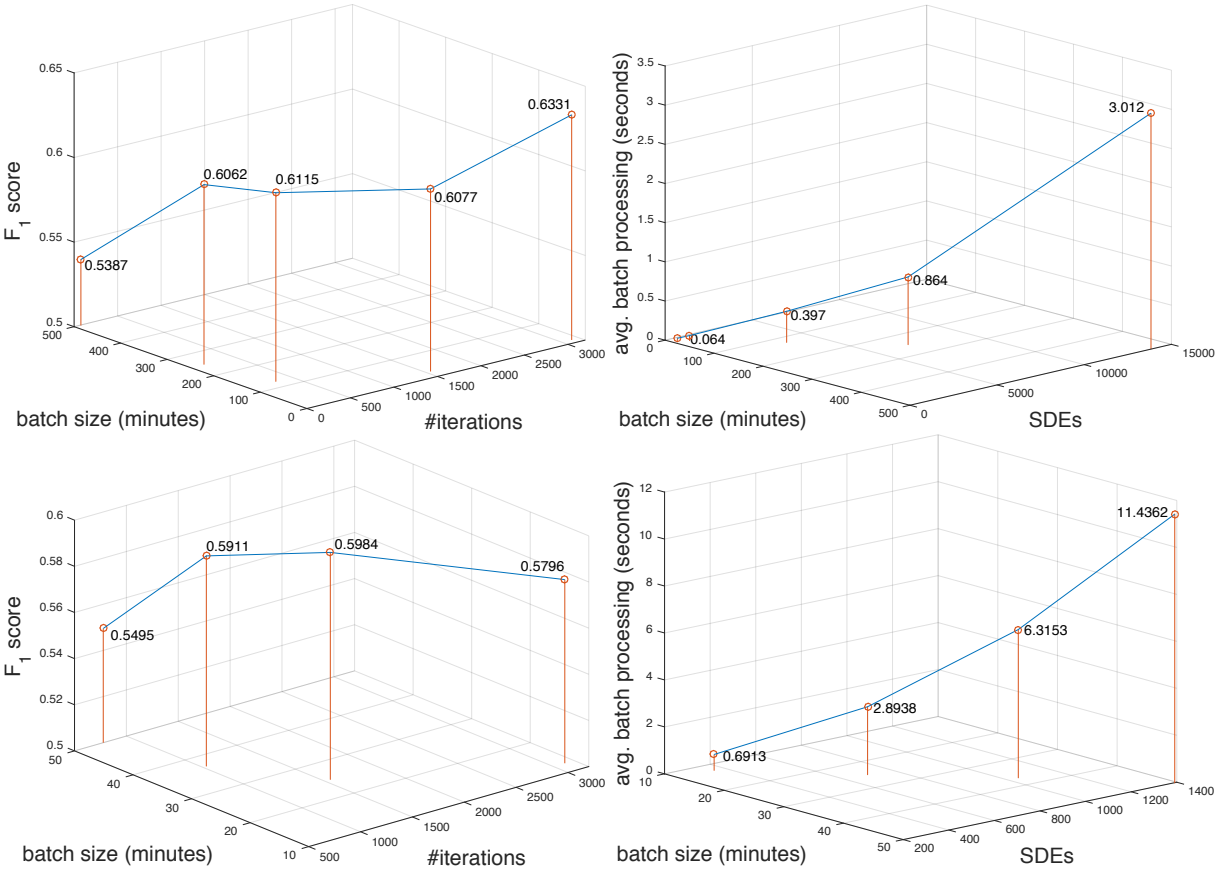

Figure 4.5: F_1 score (left) and avg. batch processing time (right) for AdaGrad (top) and OSLα (bottom). In the left figures, the Y axis shows the number of learning iterations.

4.2 Traffic Management (Simulation Rocade Data)

In this task, similarly the aim is to recognize traffic congestions which take place in the Grenoble South Ring, but instead of the real data we are using the simulated dataset provided by the CNRS. The dataset compromises 10 simulations of 1 hour duration each and a specific location is annotated with traffic congestion. The annotated location for each simulation id is different and therefore one cannot use all the simulations for learning weighted patterns for one location. Therefore, we performed experiments

for each simulation id separately. Because the training data provided for each annotated location are very little (121 timepoints), we performed training using a 60% of timepoints for training and the rest for testing. The pair of rules trained for location 1311 are presented in equation 4.5 and 4.6.

$$\texttt{InitiatedAt}(\texttt{trafficjam}(1311), t) \Leftarrow \texttt{HappensAt}(\texttt{aggr}(1311, avgspd), t) \wedge 0 \leq avgspd < 18 \tag{4.5}$$

$$\texttt{TerminatedAt}(\texttt{trafficjam}(1311), t) \Leftarrow \texttt{HappensAt}(\texttt{aggr}(1311, avgspd), t) \wedge avgspd > 45 \tag{4.6}$$

Figure 4.6 presents the evaluation results for OSLα for simulation id 1 and location 1311. The purpose of the experiment is to present the accuracy change in a fully supervised dataset. The predictive accuracy of the learned model remains almost the same as the iterations increase and arrives at the best F_1 score when batch sizes of 2.5 minutes are used. Note that the recognition results are much more accurate (highest F_1 score) than the real data case presented in Section 4.1, which arise from the fully supervised nature of the data. Finally, as in the case of the real data, OSLα can process data batches efficiently. For example, it takes ≈ 3 seconds to process a 25 minute batch (6000 SDEs).

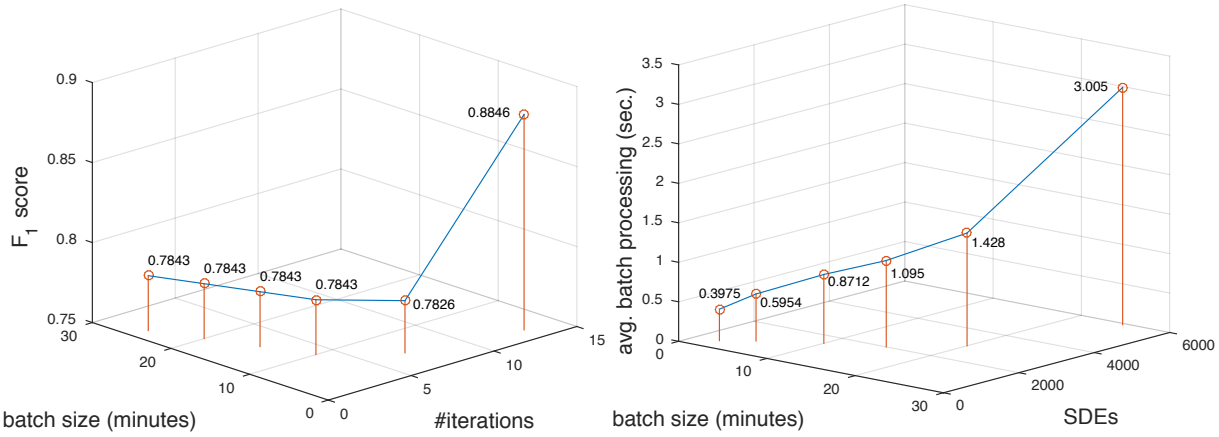

Figure 4.6: F_1 score (left) and avg. batch processing time (right) for AdaGrad (top) and OSLα (bottom). In the left figures, the Y axis shows the number of learning iterations.

4.3 Activity Recognition

In this task, the aim is to recognise complex activities that take place between multiple persons, by exploiting information about simple observed individual activities. This dataset comprises 28 surveillance videos, where each frame is annotated by human experts from the CAVIAR team on two levels. The first level contains simple, derived events (SDEs) that concern activities of individual persons or the state of objects. The second level contains composite event (CE) annotations, describing the activities between multiple persons and/or objects – i.e., people meeting and moving together, leaving an object and fighting. In Section 4.3.1 we briefly describe the dataset characteristics and the experimental setup for all of our experiments. Then in Section 4.3.2 we give an overview of the compared methods. Finally, in Sections 4.3.3 and 4.3.4 we present the results of weight learning and OSLα learning performance respectively.

4.3.1 Experimental Setup

The input to the learning methods is a stream of SDEs along with the CE annotations. The SDEs are representing people walking, running, staying active, or inactive. The first and the last time that a person or an object is tracked are represented by the SDEs enter and exit. Additionally, the coordinates of tracked persons or objects are also taken into consideration in order to express qualitative spatial relations, e.g. two persons being relatively close to each other. The CE supervision indicates when each of the CE holds. Table 4.1 presents the structure of the training sequences. Each sequence is composed of input SDEs (ground `HappensAt`), precomputed spatial constraints between pairs of people (ground `Close`), as well as the corresponding CE annotations (ground `HoldsAt`). Negated predicates in the training sequence state that the truth value of the corresponding predicate is False.

Simple Derived Events	Supervision of Composite Events
...	...
`HappensAt`(`walking`(ID_1), 100)	
`HappensAt`(`walking`(ID_2), 100)	
`OrientationMove`(ID_1, ID_2, 100)	
`Close`(ID_1, ID_2, 24, 100)	
`Next`(100, 101)	`HoldsAt`(`move`(ID_1, ID_2), 101)
...	...
`HappensAt`(`exit`(ID_1), 200)	
`HappensAt`(`walking`(ID_2), 200)	
¬`OrientationMove`(ID_1, ID_2, 200)	
¬`Close`(ID_1, ID_2, 24, 200)	
`Next`200, 201)	¬`HoldsAt`(`move`(ID_1, ID_2), 201)

Table 4.1: Training example for `move` CE. The first column is composed of a narrative of SDEs, while the second column contains the CE annotation in the form of ground `HoldsAt` predicates.

The definitions of the CEs `meet` and `move` we are using were developed in Artikis et al. (2010b). These definitions take the form of common sense rules and describe the conditions under which a CE starts or ends (`InitiatedAt`, `TerminatedAt`). For example, when two persons are walking together with the same orientation, then moving starts being recognized. Similarly, when the same persons walk away from each other, then moving stops being recognized.

From the 28 videos of the CAVIAR dataset, we have extracted 19 sequences that are annotated with the `meet` and/or `move` CEs. The rest of the sequences in the dataset are ignored, as they do not contain positive examples of the target CE. Out of the 19 sequences, 8 are annotated with both `meet` and `move` activities, 9 are annotated only with `move` and 2 only with `meet`. The total length of the extracted sequences is 12869 frames. Each frame is annotated with the (non-)occurrence of a CE and is considered an example instance. The whole dataset contains a total of 25738 annotated example instances. There are 6272 example instances in which `move` occurs and 3722 in which `meet` occurs. Consequently, for both CEs the number of negative examples is significantly larger than the number of positive examples, specifically 19466 for `move` and 22016 for `meet`. The input consists of a sequence of SDEs, i.e., `active`, `inactive`, `walking`, `running`, `enter` and `exit`, the spatial constraints `Close` and `OrientationMove`, which was precomputed, and its truth value was provided as input, as well as the supervision for `meet`

and move.

Following the work of Skarlatidis (2014); Skarlatidis et al. (2015a) we used three different inertia settings (HI, $\texttt{SI}^{\texttt{h}}$ and SI, see Table 4.2 for a description). In all three variants, all rules are always soft-constrained, except the inertia rules that, depending on the setting, may be either soft-or hard-constrained.

Settings	Description
HI	All inertia rules are hard-constrained.
$\texttt{SI}^{\texttt{h}}$	The inertia rules of HoldsAt are soft-constrained, while the rest remains hard-constrained.
SI	All inertia rules are soft-constrained.

Table 4.2: Variants of CAVIAR, using hard and soft inertia rules.

Throughout the experimental analysis, the evaluation results was obtained using the MAP inference of Huynh and Mooney (2009) and are presented in terms of True Positives (TP), False Positives (FP), False Negatives (FN), Precision, Recall and F_1 score. All reported experiment statistics are micro-averaged over the instances of recognized CEs using 10-fold cross validation over the 19 sequences. Details about the dataset are presented in Table 4.3. The experiments were performed in a computer having an Intel i7 4790@3.6GHz processor (4 cores and 8 threads) and 16GiB of RAM, running OSX El Capitan 10.11.

total SDEs	63147
average SDEs per fold	56832
total meet positive CEs	3722
total move positive CEs	6272
average meet positive CEs	3350
average move positive CEs	5600

Table 4.3: CAVIAR statistics

4.3.2 The Methods Being Compared

We begin by evaluating the online weight learning methods using manual definitions. We compare their results against the batch max-margin learning method. Moreover, for comparison purposes, we also include in the experiments the results of the logic-based activity recognition method of Artikis et al. (2010b), which we call here $\texttt{EC}_{\texttt{crisp}}$. The latter cannot perform probabilistic reasoning. Then we present the experimental results of our OSLα learning and compare them against the results obtained by the manual definitions trained by the online weight learning and the pure logic-based method $\texttt{EC}_{\texttt{crisp}}$. Finally, we also present the running times of OSL for the meet CE.

4.3.3 Weight Learning Performance

In this section both online max-margin (CDA) and AdaGrad algorithms presented in Section 3.2 are compared for all three inertia configurations of Table 4.2 against the batch max-margin learning and

$\texttt{EC}_{\texttt{crisp}}$. The CE definitions of `meet` and `move` CEs are transformed by using the domain-independent axioms of MLN–EC and exploiting the equivalences resulting from predicate completion to eliminate `InitiatedAt`, `TerminatedAt` predicates. Then by applying CNF transformation each theory results into 23 and 44 clauses for `meet` and `move` respectively. Results of weight learning for both CEs are presented in Tables 4.4 and 4.5. As expected the batch max-margin weight learning yields the best overall accuracy due the fact that it uses all the data at once in order to estimate the optimal weights. AdaGrad is the second best choice as it yields more accurate results as opposed to CDA and $\texttt{EC}_{\texttt{crisp}}$ in the majority of the cases.

Note that the $\texttt{SI}^{\texttt{h}}$ inertia setting yields the best results for each individual method. In this setting, the inertia rule of `HoldsAt` remains soft-constrained. As a result, the probability of a CE tends to decrease, even when the required termination conditions are not met. Therefore, if the probability cannot be reinforced by a re-initiation then $\texttt{SI}^{\texttt{h}}$ setting is very useful (Skarlatidis, 2014; Skarlatidis et al., 2015a). This dataset in particular has overlaps where `meet` CE has been initiated and not terminated and then `move` CE is recognized. Thereafter all occurring SDEs during the overlap are irrelevant to `meet` and cannot cause any re-initiation or termination.

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	3099	1413	523	0.6868	**0.8556**	0.7620
$\texttt{CDA}_{\texttt{HI}}$	1868	106	1754	**0.9463**	0.5157	0.6676
$\texttt{CDA}_{\texttt{SI}^{\texttt{h}}}$	1767	183	1855	0.9061	0.4878	0.6342
$\texttt{CDA}_{\texttt{SI}}$	1552	355	2070	0.8138	0.4284	0.5614
$\texttt{AdaGrad}_{\texttt{HI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{AdaGrad}_{\texttt{SI}^{\texttt{h}}}$	3096	1187	526	0.7228	0.8547	0.7833
$\texttt{AdaGrad}_{\texttt{SI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{MaxMargin}_{\texttt{HI}}$	3099	1397	523	0.6892	**0.8556**	0.7634
$\texttt{MaxMargin}_{\texttt{SI}^{\texttt{h}}}$	2946	260	676	0.9189	0.8133	**0.8629**
$\texttt{MaxMargin}_{\texttt{SI}}$	3009	809	613	0.7673	0.4770	0.5883

Table 4.4: Weight learning accuracy of the `meet` CE

Table 4.6 present the running times for learning the weights of the manual definitions. Both training and testing times are accumulated across the 10 folds. The results justify the usage of an online weight learner. Both CDA and AdaGrad are an order magnitude faster during training in contrast to the batch max-margin learner. This arises from the fact that batch learning methods require to run inference over the whole dataset in each iteration. CDA is a little faster than AdaGrad in most cases, but it lacks in accuracy and therefore AdaGrad is preferred. Moreover, AdaGrad uses L1-regularization in contrast to CDA which uses L2 and is more appropriate for the task of structure learning, which may introduce a lot of new features and many of which may not be useful. Note that testing times are almost identical because the set of learned clauses used as an input to these methods is the same and only their resulting weights differ. Therefore grounding and inference times do not change substantially. Consequently, the presented experimental evaluation justifies our choice for selecting AdaGrad in our online structure learning method.

Figure 4.7 present the F1 score change as batch size increases. Note that the accuracy of the model increases in most cases with respect to batch size. Nevertheless, the batch size can lower the accuracy (see `meet` CE for batch size 1000) if the batches bisect SDEs depending on each other for correctly

Method	TP	FP	FN	Precision	Recall	F_1 score
EC_{crisp}	4008	400	2264	0.9093	0.6390	0.7506
CDA_{HI}	4008	384	2264	0.9125	0.6390	0.7516
CDA_{SI^h}	2437	261	1197	0.9032	0.6706	0.7697
CDA_{SI}	3386	368	2886	0.9019	0.5398	0.6754
$AdaGrad_{HI}$	4008	384	2264	0.9125	0.6390	0.7516
$AdaGrad_{SI^h}$	4077	368	2031	**0.9172**	0.6674	0.7726
$AdaGrad_{SI}$	3148	1542	3124	0.6712	0.5019	0.5743
$MaxMargin_{HI}$	5598	1128	674	0.8323	0.8925	0.8614
$MaxMargin_{SI^h}$	5902	1088	370	0.8443	**0.9410**	**0.8901**
$MaxMargin_{SI}$	3226	1268	1760	0.6599	0.6470	0.6534

Table 4.5: Weight learning accuracy of the move CE

Method	meet			move		
	total	training	testing	total	training	testing
CDA_{HI}	16m 29s	14m 40s	1m 49s	27m 20s	24m 26s	2m 54s
CDA_{SI^h}	**15m 52s**	**14m 1s**	1m 51s	**19m 23s**	**17m 15s**	2m 8s
CDA_{SI}	15m 54s	14m 18s	1m 36s	25m 55s	23m 44s	2m 11s
$AdaGrad_{HI}$	18m 28s	16m 37s	1m 51s	26m 29s	23m 31s	2m 58s
$AdaGrad_{SI^h}$	16m 2s	14m 16s	1m 46s	28m 17s	25m 48s	2m 29s
$AdaGrad_{SI}$	**15m 52s**	14m 20s	1m 32s	32m 10s	29m 39s	2m 31s
$MaxMargin_{HI}$	3h 12m 29s	3h 10m 41s	1m 48s	4h 28m 27s	4h 26m 2s	2m 25s
$MaxMargin_{SI^h}$	3h 10m 41s	3h 8m 47s	1m 54s	4h 42m 36s	4h 41m 38s	1m 58s
$MaxMargin_{SI}$	3h 27m 32s	3h 25m 54s	1m 38s	4h 58m 34s	4h 56m 27s	2m 7s

Table 4.6: Weight learning running times for meet and move CE

recognising a CE into different batches. Figure 4.8 present the average training time per fold as batch size increases. Note that the training time increases linearly with the batch size.

4.3.4 OSLα Performance

In this section the experiments for our online probabilistic structure learning method are presented. The model is trained discriminitavely and its performance is evaluated using 10-fold cross-validation over 19 sequences. The results are compared against the results presented in the previous Section and OSL. In order to achieve the best possible accuracy for OSLα, tuning of the evaluation threshold μ (see Section 3.4.4) is required. We run structure learning using 10-fold cross validation over 5 distinct values of μ. The results are presented in Figure 4.9. The highest accuracy is achieved by using μ=4 and μ=1 for meet and move CEs respectively. In the case of meet the evaluation threshold smoothly affects accuracy, in contrast to move where a lot of important structures are penalized if OSLα becomes more austere and accuracy is decreasing significantly.

Then by using the best thresholds detailed results of OSLα over 10 folds are presented in Tables

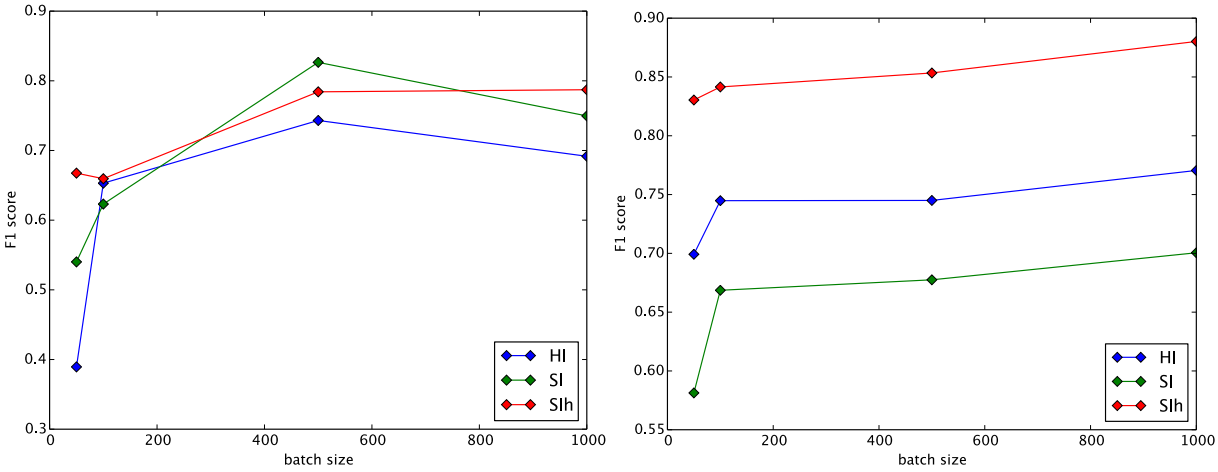

Figure 4.7: F1 score change as batch size increases for `meet` (left) and `move`(right).

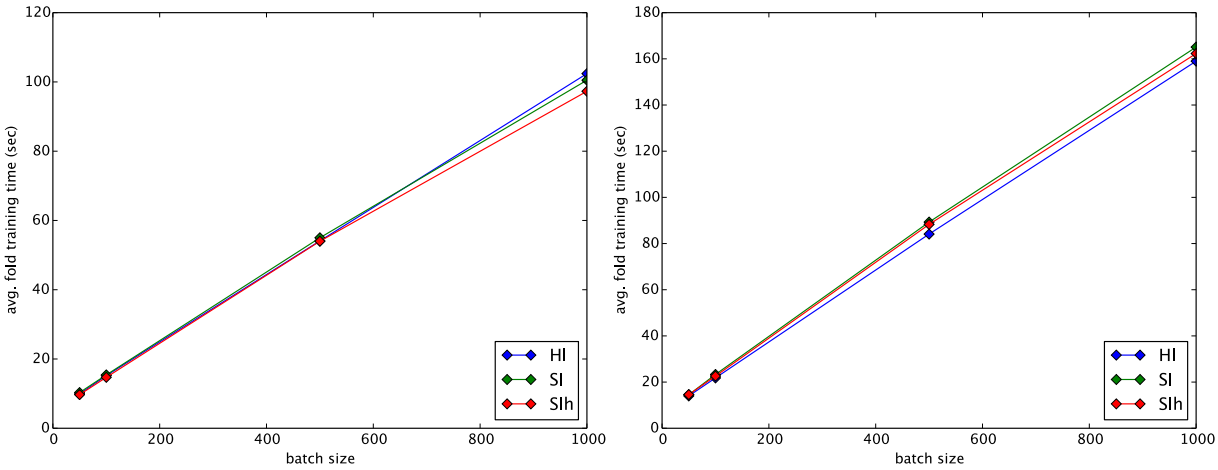

Figure 4.8: Average training time per fold as batch size increases for `meet` (left) and `move`(right). Average number of SDEs for batch sizes 50, 100, 500, and 1000 are 338, 688, 3244, and 5840 respectively.

4.7a and 4.7b. Note that OSLα can be at least as accurate as AdaGrad training definitions developed by hand or even better (see Tables 4.4 and 4.5 for comparison). Different inertia configurations do not affect the accuracy. That is happening because the dataset contains SDEs in every single timepoint and the definitions learned by OSLα introduce all possible predicates in the bodies of the template atoms (i.e. `InitiatedAt`,`TerminatedAt`) due to the exhaustive hypergraph search. As a consequence, inertia rules are outweighed by the rest of the theory.

Furthermore, Table 4.8 present the running times of OSLα for both CEs. Both training and testing times are accumulated across the 10 folds. Training time of `move` is much slower than the one of `meet`. That is because `move` includes another predicate (`OrientationMove`) in its predicate schema and therefore yields more possible structures. Moreover testing times, although pretty fast, still is slower in contrast to the ones presented in the weight learning experiments (see Table 4.6). However, structure learning yields a lot of clauses and therefore the grounding of the theory during inference is slower.

Then we attempted to perform probabilistic structure learning on this dataset using OSL. Specifically, we began running experiments for the `meet` CE and we terminated the experimentation after 25 hours. During this time OSL had processed only 4 training examples (micro-batches) out of the 17 con-

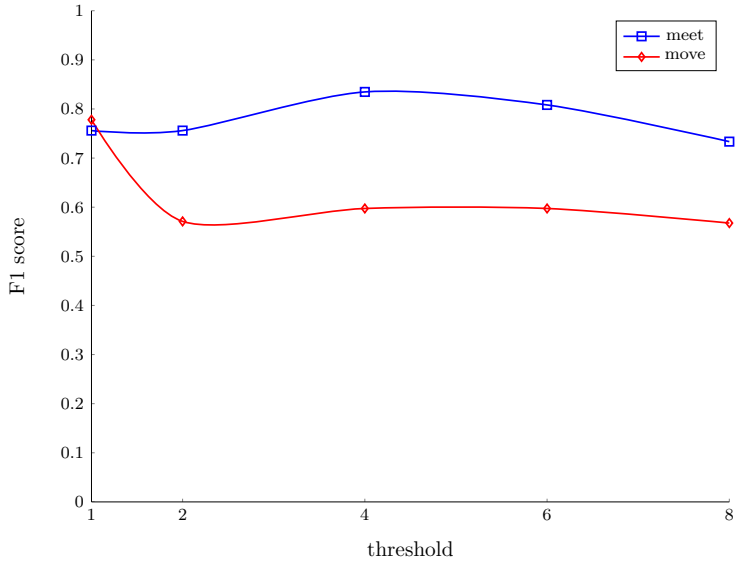

Figure 4.9: F1 score over 10 folds for various evaluation threshold values.

tained in the first fold. OSLα on the other hand performed 10 fold cross validation for the `meet` CE in about 4 hours.

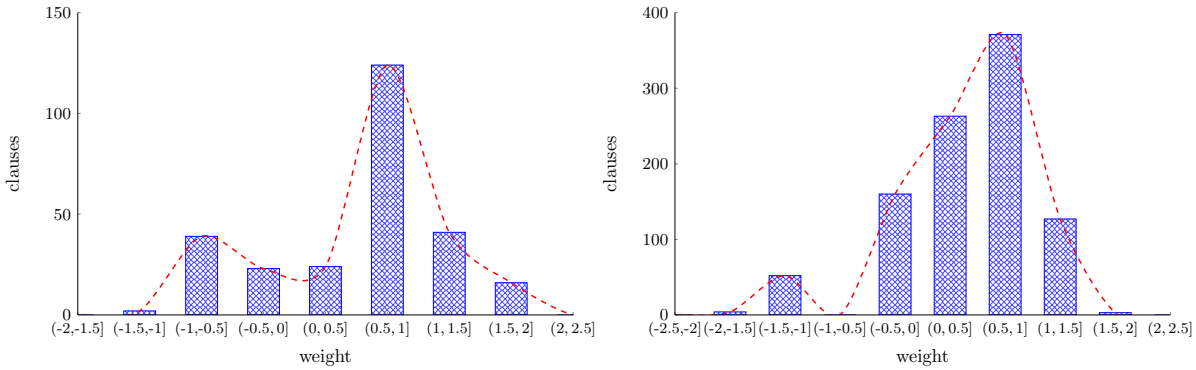

Figure 4.10: Weight distibution learned for `meet` (left) and `move` (right)

For effective CE recognition, we prune a portion of the learned weighted structures having weights below a certain threshold θ (see Section 3.4.4), for various values of θ, and present the results in terms of both accuracy and testing time. We begin by running OSLα using all the 19 sequences of the dataset and present a histogram for each CE representing the distribution of weights learned. The distributions are shown in Figure 4.10. The presented histogram give an insight of the portion of the theory that is about to be pruned for each chosen θ value.

Note that there is a considerable amount of clauses having significantly small weight values. These clauses may be pruned in order to simplify the resulting model without significantly affecting the accuracy, but yielding better inference times. Therefore, we prune the resulting structure for 3 distinct values of θ and present the change in the behavior with respect to the original model, where all learned structures were retained (θ=0). The following results are obtained over 10 folds. Figure 4.11 present the reduction in the number of clauses in the resulting theory as θ increases. This reduction corresponds to missing features that can affect accuracy and on the other hand reduce the inference time.

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	3099	1413	523	0.6868	**0.8556**	0.7620
$\texttt{AdaGrad}_{\texttt{SI}^\texttt{h}}$	3096	1187	526	0.7228	0.8547	0.7833
$\texttt{MaxMargin}_{\texttt{SI}^\texttt{h}}$	2946	260	676	**0.9189**	0.8133	**0.8629**
$\texttt{OSL}\alpha_{\texttt{HI}}$	3082	680	540	0.8192	0.8509	0.8347
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	3082	680	540	0.8192	0.8509	0.8347
$\texttt{OSL}\alpha_{\texttt{SI}}$	3082	680	540	0.8192	0.8509	0.8347

(a) Results for the `meet` CE

Method	TP	FP	FN	Precision	Recall	F_1 score
$\texttt{EC}_{\texttt{crisp}}$	4008	400	2264	0.9093	0.6390	0.7506
$\texttt{AdaGrad}_{\texttt{SI}^\texttt{h}}$	4077	368	2031	**0.9172**	0.6674	0.7726
$\texttt{MaxMargin}_{\texttt{SI}^\texttt{h}}$	5902	1088	370	0.8443	**0.9410**	**0.8901**
$\texttt{OSL}\alpha_{\texttt{HI}}$	4718	1138	1554	0.8056	0.7522	0.7780
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	4718	1138	1554	0.8056	0.7522	0.7780
$\texttt{OSL}\alpha_{\texttt{SI}}$	4718	1138	1554	0.8053	0.7522	0.7779

(b) Results for the `move` CE

Table 4.7: Results for $\texttt{OSL}\alpha$ for μ= 4 and μ=1 respectively.

Method	`meet`			`move`		
	total	training	testing	total	training	testing
$\texttt{OSL}\alpha_{\texttt{HI}}$	**3h 57m 12s**	**3h 48m 13s**	8m 59s	**19h 31m 29s**	**19h 20m 24s**	11m 5s
$\texttt{OSL}\alpha_{\texttt{SI}^\texttt{h}}$	3h 58m 58s	3h 50m 43s	8m 15s	20h 0m 50s	19h 50m 58s	9m 52s
$\texttt{OSL}\alpha_{\texttt{SI}}$	4h 1m 5s	3h 52m 52s	9m 13s	19h 58m 43s	19h 48m 23s	10m 20s

Table 4.8: $\texttt{OSL}\alpha$ running times for `meet` and `move` CE

Figures 4.12 and 4.13 present the effect of θ in accuracy and testing time. Note that θ=0.5 results in a slight reduction in accuracy for `move` and no reduction for `meet`, but testing time is much better. Therefore, we can prune a subset of the resulting theory using an optimal value for θ in order to achieve a minimal set of clauses yielding the best overall accuracy and inference performance.

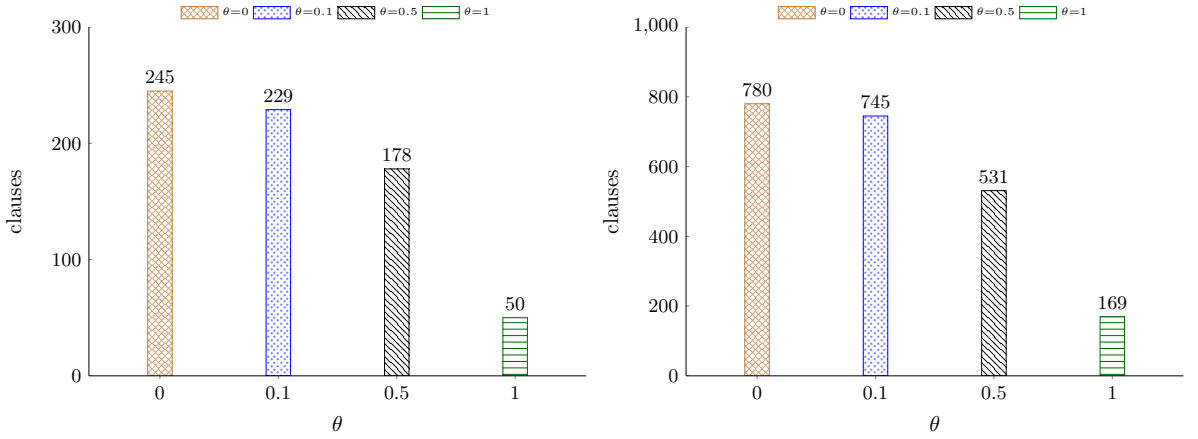

Figure 4.11: Reduction in the number of clauses learned as θ increases for `meet` (left) and `move` (right).

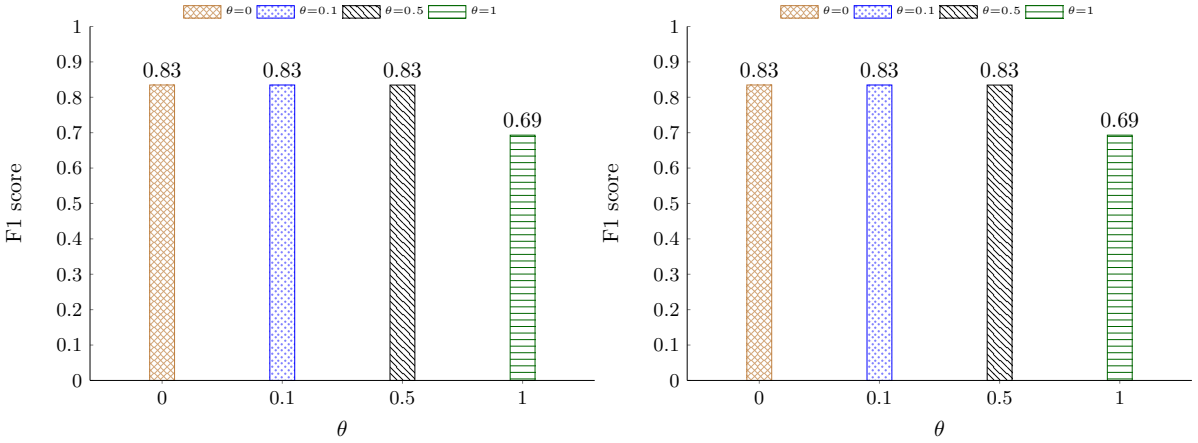

Figure 4.12: Effect of F1 score as θ increases for `meet` (left) and `move` (right).

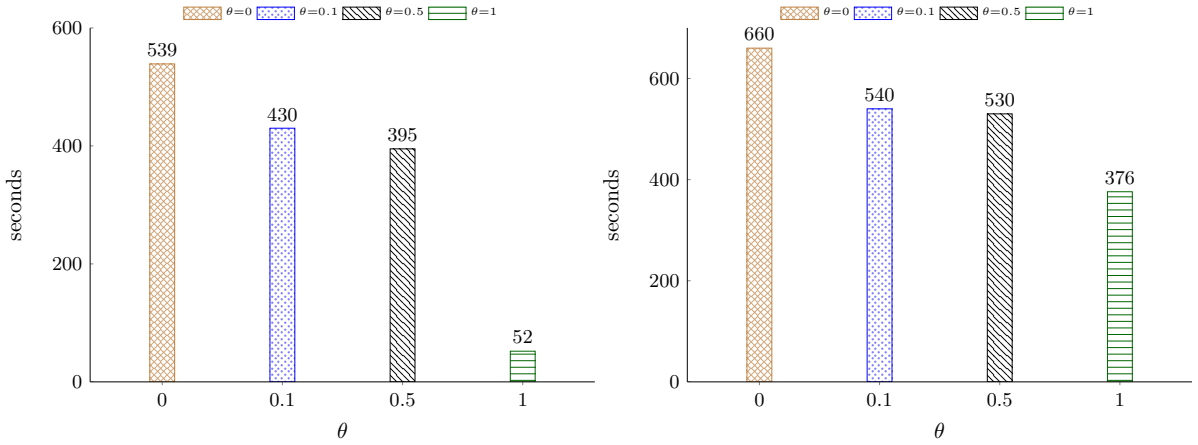

Figure 4.13: Reduction of testing time as θ increases for `meet` (left) and `move` (right).

5

Conclusions

We presented a survey of methods for complex event recognition under uncertainty, i.e. in cases where the input stream may be probabilistic and/or in cases where the rules for detecting complex events may be characterized by a lack of absolute confidence. The two dominant approaches are those that employ probabilistic/stochastic automata and those that are logic-based. A small number of research efforts have employed other frameworks in order to address other issues. For example, probabilistic Petri Nets, which are powerful when modeling concurrency and synchronization of activities, and stochastic context-free grammars, with which it is easy to model hierarchies of events and recursive patterns. However, these frameworks have not been systematically explored and general conclusions are not easy to be drawn. In general, they lack the means to express relational structures and are not suited for event recognition. As far as the two dominant classes of methods are concerned, our survey has shown that a trade-off emerges between complexity and performance. On the one hand, the logic-based systems are expressive and can model complex probabilistic , but suffer from under-performance. On the other hand, automata-based systems yield high performance with respect to throughput, but with simple patterns and dependencies.

We also presented our approach for scalable incremental learning of event definitions from large amounts of data. As far as parameter learning is concerned, we described our algorithm of choice (AdaGrad), whose main feature is that it gives frequently occurring features very low learning rates and infrequent features high learning rates. This way, finding and identifying predictive but comparatively rare features becomes easier. With respect to parameter learning, we presented the state-of-the-art methods and identified their weaknesses. We noted that they are batch learning algorithms, effectively designed for training data consisting of relatively few mega-examples. Moreover, most of these algorithms are strictly data-driven and they do not exploit background knowledge, possibly spending a lot of time exploring clauses that are true in most of the possible worlds, therefore largely useless for purposes of learning. We subsequently described our preferred algorithm that attempts to overcome these limitations when learning definitions for complex events. It is a variant of the Online Structure Learning algorithm, which takes into account the predicted possible worlds, i.e. the most probable possible worlds predicted by the current model. Our variant exploits domain-independent axioms defined in the knowledge base, to further constrain the search space to clauses subject to specific characteristics of the Event Calculus formalism that we use for event recognition.

Finally, we presented two sets of experiments for the learning algorithms. We tested our parameter learning algorithm, along with other alternatives, against a publicly available video dataset, showing that

AdaGrad indeed strikes a balance between performance and accuracy. We subsequently used AdaGrad for weight learning of traffic jam rules with the dataset provided by CNRS for the Grenoble South Ring. We showed that if the rules are specialized against location and lane type, their weights show much less fluctuations and are more stable, compared to more generic rules, even when annotation is not ideal.

In the future, we aim to build upon our survey work on probabilistic event recognition in order to develop a forecasting system. Moreover, our goal is to refine the rules for the traffic use case, according to the results obtained from our learning system. Additionally, we will apply this same system on the other use case of credit card fraud detection. As another step, our structure learning algorithm will be applied against both datasets.

Bibliography

J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD Conference*, page 147160, 2008.

M. Albanese, V. Moscato, A. Picariello, V. S. Subrahmanian, and O. Udrea. Detecting Stochastically Scheduled Activities in Video. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1802–1807, 2007.

M. Albanese, R. Chellappa, N. P. Cuntoor, V. Moscato, A. Picariello, V. S. Subrahmanian, and O. Udrea. A Constrained Probabilistic Petri Net Framework for Human Activity Detection in Video. *IEEE Transactions on Multimedia*, 10(6):982–996, 2008.

M. Albanese, R. Chellappa, N. Cuntoor, V. Moscato, A. Picariello, V. S. Subrahmanian, and O. Udrea. PADS: A Probabilistic Activity Detection Framework for Video Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2246–2261, 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2010.33.

M. Albanese, C. Molinaro, F. Persia, A. Picariello, and V. S. Subrahmanian. Finding "Unexplained" Activities in Video. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1628–1634, 2011.

J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983a.

J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832843, 1983b.

J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, July 1984. ISSN 0004-3702.

A. Artikis, M. Sergot, and G. Paliouras. A Logic Programming Approach to Activity Recognition. In *Proceedings of the 2nd International Workshop on Events in Multimedia (EiMM)*, pages 3–8. ACM, 2010a.

A. Artikis, A. Skarlatidis, and G. Paliouras. Behaviour Recognition from Video Content: a Logic Programming Approach. *International Journal on Artificial Intelligence Tools (JAIT)*, 19(2):193–209, 2010b.

A. Artikis, O. Etzion, Z. Feldman, and F. Fournier. Event processing under uncertainty. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS '12, pages 32–43, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1315-5.

M. Biba, S. Ferilli, and F. Esposito. Discriminative structure learning of markov logic networks. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, ILP '08, pages 59–76, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85927-7. doi: 10.1007/978-3-540-85928-4_9. URL http://dx.doi.org/10.1007/978-3-540-85928-4_9.

R. Biswas, S. Thrun, and K. Fujimura. Recognizing Activities with Multiple Cues. In *Proceedings of the 2nd Workshop on Human Motion - Understanding, Modeling, Capture and Animation*, Lecture Notes in Computer Science, pages 255–270. Springer, 2007.

H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1–2):285–297, 1998. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(98)00034-4. URL http://www.sciencedirect.com/science/article/pii/S0004370298000344.

M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, page 994999. IEEE Computer Society, 1997.

W. Brendel, A. Fern, and S. Todorovic. Probabilistic Event Logic for Interval-based Event Recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3329–3336. IEEE Computer Society, 2011.

H. V. Bromberg F, Margaritis D. Efficient markov network structure discovery using independence tests. *The journal of artificial intelligence research*, 35(1):449–484, 2009.

M. Bruynooghe, B. De Cat, J. Drijkoningen, D. Fierens, J. Goos, B. Gutmann, A. Kimmig, W. Labeeuw, S. Langenaken, N. Landwehr, W. Meert, E. Nuyts, R. Pellegrims, R. Rymenants, S. Segers, I. Thon, J. Van Eyck, G. Van den Broeck, T. Vangansewinkel, L. Van Hove, J. Vennekens, T. Weytjens, and L. De Raedt. An exercise with statistical relational learning systems, July 2009.

I. Cervesato and A. Montanari. A calculus of macro-events: progress report. In *Seventh International Workshop on Temporal Representation and Reasoning, 2000. TIME 2000. Proceedings*, pages 47–58, 2000.

X. Chuanfei, L. Shukuan, W. Lei, and Q. Jianzhong. Complex event detection in probabilistic stream. In *Web Conference (APWEB), 2010 12th International Asia-Pacific*, pages 361–363, Apr. 2010.

R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387718230, 9780387718231.

G. Cugola and A. Margara. TESLA: a formally defined event specification language. In *Proceedings of Conference on Distributed-Event Based Systems (DEBS)*, page 5061, 2010.

G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, 44(3), 2011.

G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli. Introducing uncertainty in complex event processing: model, implementation, and validation. *Computing*, pages 1–42, 2014. ISSN 0010-485X. doi: 10.1007/s00607-014-0404-y. URL http://dx.doi.org/10.1007/s00607-014-0404-y.

J. Davis and P. Domingos. Bottom-up learning of markov network structure. In *In Proc. of the 27th International Conference on Machine Learning*. ACM Press, 2010.

L. De Raedt and L. Dehaspe. Clausal discovery. *Mach. Learn.*, 26(2-3):99–146, Mar. 1997. ISSN 0885-6125. doi: 10.1023/A:1007361123060. URL http://dx.doi.org/10.1023/A:1007361123060.

L. De Raedt and K. Kersting. Probabilistic logic learning. *SIGKDD Explor. Newsl.*, 5(1):31–48, July 2003. ISSN 1931-0145. doi: 10.1145/959242.959247.

A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In Y. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, editors, *Advances in Database Technology - EDBT 2006*, number 3896 in Lecture Notes in Computer Science, pages 627–644. Springer Berlin Heidelberg, Jan. 2006.

Q.-T. Dinh, M. Exbrayat, and C. Vrain. Heuristic method for discriminative structure learning of markov logic networks. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 163–168, December 2010. doi: 10.1109/ICMLA.2010.31.

P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

C. Dousson. Extending and unifying chronicle representation with event counters. In *ECAI*, pages 257–261. Citeseer, 2002.

C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI*, volume 7, pages 324–329, 2007.

C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition: representation and algorithms. In *IJCAI*, volume 93, pages 166–172, 1993.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953048.2021068.

O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010a. ISBN 978-1-935182-21-4.

O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications Company, 2010b. ISBN 978-1-935182-21-4.

R. Fagin. Combining fuzzy information from multiple systems. In *In Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 216–226. ACM Press, 1996. ISBN 0-89791-781-2.

D. Fierens, G. Van den Broeck, J. Renkens, D. Shterionov, B. Gutmann, I. Thon, G. Janssens, and L. De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, pages 1–44, 2013.

N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1300–1307, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1624312.1624404.

N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, Jan. 1997. ISSN 1046-8188.

A. Gal, S. Wasserkrug, and O. Etzion. Event Processing over Uncertain Data. In S. Helmer, A. Poulovassilis, and F. Xhafa, editors, *Reasoning in Event-Based Distributed Systems*, volume 347 of *Studies in Computational Intelligence*, pages 279–304. Springer, 2011. ISBN 978-3-642-19723-9.

L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007. ISBN 9780262072885.

M. L. Ginsberg. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

S. Gong and T. Xiang. Recognition of group activities using dynamic probabilistic networks. In *Proceedings of the 9th International Conference on Computer Vision (ICCV)*, volume 2, page 742749. IEEE Computer Society, 2003.

T. Grabs and M. Lu. Measuring performance of complex event processing systems. In *Topics in Performance Evaluation, Measurement and Characterization*, pages 83–96. Springer, 2012.

E. Hazan, A. Kalai, S. Kale, and A. Agarwal. Logarithmic regret algorithms for online convex optimization. In *In 19th COLT*, pages 499–513, 2006.

D. Heckerman. Learning in graphical models. chapter A Tutorial on Learning with Bayesian Networks, pages 301–354. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-60032-3. URL http://dl.acm.org/citation.cfm?id=308574.308676.

R. Helaoui, M. Niepert, and H. Stuckenschmidt. Recognizing Interleaved and Concurrent Activities: A Statistical-Relational Approach. In *Proceedings of the 9th Annual International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9. IEEE Computer Society, 2011.

T. N. Huynh and R. J. Mooney. Discriminative structure and parameter learning for markov logic networks. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 416–423, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390209. URL http://doi.acm.org/10.1145/1390156.1390209.

T. N. Huynh and R. J. Mooney. Max-Margin Weight Learning for Markov Logic Networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, volume 5781 of *Lecture Notes in Computer Science*, pages 564–579. Springer, 2009.

T. N. Huynh and R. J. Mooney. Online Max-Margin Weight Learning for Markov Logic Networks. In *Proceedings of the 11th SIAM International Conference on Data Mining (SDM11)*, pages 642–651, Mesa, Arizona, USA, April 2011a.

T. N. Huynh and R. J. Mooney. Online structure learning for markov logic networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD 2011)*, volume 2, pages 81–96, September 2011b. URL http://www.cs.utexas.edu/users/ai-lab/?huynh:ecml11.

Y. Ivanov and A. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):852–872, Aug. 2000. ISSN 0162-8828. doi: 10.1109/34.868686.

M. Jaeger. Relational bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, pages 266–273, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-485-5.

M. Jaeger. Model-theoretic expressivity analysis. In L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic inductive logic programming*, number 4911 in Lecture Notes in Computer Science, pages 325–339. Springer Berlin Heidelberg, Jan. 2008.

T. Joachims. A support vector method for multivariate performance measures. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, pages 377–384. ACM Press, 2005. URL http://doi.acm.org/10.1145/1102351.1102399.

T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, 2009. ISSN 0885-6125. doi: 10.1007/s10994-009-5108-8. URL http://dx.doi.org/10.1007/s10994-009-5108-8.

A. Kanaujia, T. E. Choe, and H. Deng. Complex events recognition under uncertainty in a sensor network. *arXiv:1411.0085 [cs]*, Nov. 2014. arXiv: 1411.0085.

H. Kautz, B. Selman, and Y. Jiang. A General Stochastic Approach to Solving Problems with Hard and Soft Constraints. In D. Gu, J. Du, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 573–586. AMS, 1997.

H. Kawashima, H. Kitagawa, and X. Li. Complex event processing over uncertain data streams. In *2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 521–526, Nov. 2010. doi: 10.1109/3PGCIC.2010.89.

K. Kersting, L. de Raedt, and T. Raiko. Logical Hidden Markov Models. *Journal of Artificial Intelligence Research (JAIR)*, 25(1):425–456, 2006.

H. Khosravi, O. Schulte, T. Man, X. Xu, and B. Bina. Structure learning for markov logic networks with many descriptive attributes. In M. Fox and D. Poole, editors, *AAAI*. AAAI Press, 2010. URL http://dblp.uni-trier.de/db/conf/aaai/aaai2010.html#KhosraviSMXB10.

T. Khot, S. Natarajan, K. Kersting, and J. Shavlik. Learning markov logic networks via functional gradient boosting, 2011.

T. Khot, S. Natarajan, K. Kersting, and J. Shavlik. Gradient-based boosting for statistical relational learning: The markov logic network and missing data cases. *Mach. Learn.*, 100(1):75–100, July 2015. ISSN 0885-6125. doi: 10.1007/s10994-015-5481-4. URL http://dx.doi.org/10.1007/s10994-015-5481-4.

A. Kimmig, B. Demoen, L. D. Raedt, V. S. Costa, and R. Rocha. On the implementation of the probabilistic logic programming language ProbLog. *Theory and Practice of Logic Programming*, 11: 235262, 2011.

S. Kok and P. Domingos. Learning the structure of Markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448. ACM, 2005.

S. Kok and P. Domingos. Learning markov logic network structure via hypergraph lifting. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 505–512. ACM, 2009.

S. Kok and P. Domingos. Learning markov logic networks using structural motifs. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 551–558. Citeseer, 2010.

R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):6795, 1986.

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 282–289. Morgan Kaufmann, 2001.

G. Lavee, M. Rudzsky, and E. Rivlin. Propagating Certainty in Petri Nets for Activity Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(2):326–337, Feb 2013. ISSN 1051-8215. doi: 10.1109/TCSVT.2012.2203742.

S. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of markov networks using l1 regularization. In *In NIPS*, 2006.

L. Liao, D. Fox, and H. A. Kautz. Hierarchical conditional random fields for GPS-based activity recognition. In *International Symposium of Robotics Research (ISRR)*, volume 28 of *Springer Tracts in Advanced Robotics (STAR)*, page 487506. Springer, 2005.

H. Loureno, O. Martin, and T. Sttzle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 320–353. Springer US, 2003. ISBN 978-1-4020-7263-5. doi: 10.1007/0-306-48056-5_11. URL http://dx.doi.org/10.1007/0-306-48056-5_11.

D. Lowd and J. Davis. Learning markov network structure with decision trees. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 334–343, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4256-0. doi: 10.1109/ICDM.2010.128. URL http://dx.doi.org/10.1109/ICDM.2010.128.

D. Lowd and J. Davis. Improving markov network structure learning using decision trees. *J. Mach. Learn. Res.*, 15(1):501–532, Jan. 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2627451.

D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. ISBN 0201727897.

J. Ma, W. Liu, and P. Miller. Event modelling and reasoning with uncertain information for distributed sensor networks. In *Scalable Uncertainty Management*, pages 236–249. Springer, 2010.

C. Manfredotti. Modeling and Inference with Relational Dynamic Bayesian Networks. In Y. Gao and N. Japkowicz, editors, *Advances in Artificial Intelligence*, volume 5549 of *Lecture Notes in Computer Science*, pages 287–290. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-01817-6.

C. Manfredotti, H. Hamilton, and S. Zilles. Learning RDBNs for Activity Recognition. In *NIPS Workshop on Learning and Planning from Batch Time Series Data*, 2010.

A. McCallum. Efficiently inducing features of conditional random fields. *CoRR*, abs/1212.2504, 2012. URL http://arxiv.org/abs/1212.2504.

M. R. Mendes, P. Bizarro, and P. Marques. A performance study of event processing systems. In *Performance Evaluation and Benchmarking*, pages 221–236. Springer, 2009.

M. R. Mendes, P. Bizarro, and P. Marques. Towards a standard event processing benchmark. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, page 307310, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1636-1. doi: 10.1145/2479871.2479913.

L. Mihalkova and R. Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*, pages 625–632. ACM, 2007.

D. Minnen, I. Essa, and T. Starner. Expectation grammars: leveraging high-level expectations for activity recognition. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings*, volume 2, pages II–626–II–632 vol.2, June 2003. doi: 10.1109/CVPR.2003.1211525.

C. Molinaro, V. Moscato, A. Picariello, A. Pugliese, A. Rullo, and V. S. Subrahmanian. PADUA: Parallel Architecture to Detect Unexplained Activities. *ACM Transactions on Internet Technology (TOIT)*, 14 (1):3:1–3:28, Aug. 2014. ISSN 1533-5399. doi: 10.1145/2633685. URL http://doi.acm.org/10.1145/2633685.

D. Moore and I. Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In *AAAI/IAAI*, pages 770–776, 2002.

V. I. Morariu and L. S. Davis. Multi-agent event recognition in structured scenarios. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3289–3296. IEEE Computer Society, 2011.

E. T. Mueller. Event Calculus. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 671–708. Elsevier, 2008.

S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13:245–286, 1995. doi: 10.1007/BF03037227.

S. Muggleton and J. Chen. A behavioral comparison of some probabilistic logic models. In L. D. Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic Inductive Logic Programming*, number 4911 in Lecture Notes in Computer Science, pages 305–324. Springer Berlin Heidelberg, Jan. 2008. ISBN 978-3-540-78651-1, 978-3-540-78652-8.

K. P. Murphy. *Dynamic Bayesian Networks: representation, inference and learning*. PhD thesis, University of California, 2002.

S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Mach. Learn.*, 86(1):25–56, Jan. 2012. ISSN 0885-6125. doi: 10.1007/s10994-011-5244-9. URL http://dx.doi.org/10.1007/s10994-011-5244-9.

A. Paschke. ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event/action logics and transactional update logics. *arXiv preprint cs/0610167*, 2006.

A. Paschke and M. Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, Dec. 2008. ISSN 0167-9236.

S. D. Pietra, V. D. Pietra, and J. Lafferty. Inducing features of random fields. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 19(4):380–393, 1997.

J. R. Quinlan. Learning logical definitions from relations. *MACHINE LEARNING*, 5:239–266, 1990.

L. R. Rabiner and B.-H. Juang. An introduction to Hidden Markov Models. *Acoustics, Speech, and Signal Processing Magazine (ASSP)*, 3(1):4–16, 1986.

P. Ravikumar, M. J. Wainwright, and J. D. Lafferty. High dimensional ising model selection using l1 regularized logistic regression. *Ann. Statist.*, 38(3):1287–1319, 06 2010. doi: 10.1214/09-AOS691. URL http://dx.doi.org/10.1214/09-AOS691.

C. Ré, J. Letchner, M. Balazinksa, and D. Suciu. Event queries on correlated probabilistic streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 715–728, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6. doi: 10.1145/1376616.1376688. URL http://doi.acm.org/10.1145/1376616.1376688.

B. L. Richards and R. J. Mooney. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 50–55. AAAI Press, 1992. ISBN 0-262-51063-4. URL http://dl.acm.org/citation.cfm?id=1867135.1867143.

M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(12):107136, 2006.

M. S. Ryoo and J. K. Aggarwal. Recognition of Composite Human Activities through Context-Free Grammar Based Representation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1709–1718. IEEE Computer Society, 2006.

M. S. Ryoo and J. K. Aggarwal. Semantic Representation and Recognition of Continued and Recursive Human Activities. *International Journal of Computer Vision*, 82(1):1–24, 2009.

A. Sadilek and H. A. Kautz. Location-Based Reasoning about Complex Multi-Agent Behavior. *Journal of Artificial Intelligence Research (JAIR)*, 43:87–133, 2012.

S. Sanghai, P. Domingos, and D. Weld. Relational dynamic bayesian networks. *Journal of Artificial Intelligence Research*, 24(2005):759–797, 2005.

J. Selman, M. R. Amer, A. Fern, and S. Todorovic. PEL-CNF: Probabilistic event logic conjunctive normal form for video interpretation. In *Proceedings of the International Conference on Computer Vision Workshops (ICCVW)*, pages 680–687. IEEE Computer Society, 2011.

Z. Shen, H. Kawashima, and H. Kitagawa. Probabilistic event stream processing with lineage. In *Proc. of Data Engineering Workshop*, 2008.

V. D. Shet, J. Neumann, V. Ramesh, and L. S. Davis. Bilattice-based Logical Reasoning for Human Detection. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE Computer Society, 2007.

V. D. Shet, M. Singh, C. Bahlmann, V. Ramesh, J. Neumann, and L. S. Davis. Predicate Logic Based Image Grammars for Complex Pattern Recognition. *International Journal of Computer Vision*, 93(2): 141–161, 2011.

J. M. Siskind. Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic. *Journal of Artificial Intelligence Research (JAIR)*, 15:3190, 2001.

A. Skarlatidis. *Event Recognition Under Uncertainty and Incomplete Data*. PhD thesis, Department of Digital Systems, University of Piraeus, October 2014.

A. Skarlatidis, G. Paliouras, G. Vouros, and A. Artikis. Probabilistic event calculus based on markov logic networks. In *Proceedings of the 5th International Symposium on Rules (RuleML)*, volume 7018 of *Lecture Notes in Computer Science*, page 155170. Springer, 2011.

A. Skarlatidis, A. Artikis, J. Filippou, and G. Paliouras. A probabilistic logic programming event calculus. *Journal of Theory and Practice of Logic Programming (TPLP)*, 2013.

A. Skarlatidis, E. Michelioudakis, N. Katzouris, A. Artikis, G. Paliouras, E. Alevizos, I. Vetsikas, and C. Vlassopoulos. Event recognition and forecasting technology (part 2), 02 2014.

A. Skarlatidis, G. Paliouras, A. Artikis, and G. A. Vouros. Probabilistic Event Calculus for Event Recognition. *ACM Transactions on Computational Logic*, 16(2):11:1–11:37, Feb. 2015a. ISSN 1529-3785. doi: 10.1145/2699916. URL http://doi.acm.org/10.1145/2699916.

A. Skarlatidis, G. Paliouras, A. Artikis, and G. A. Vouros. Probabilistic event calculus for event recognition. *ACM Trans. Comput. Logic*, 16(2):11:1–11:37, Feb. 2015b. ISSN 1529-3785. doi: 10.1145/2699916. URL http://doi.acm.org/10.1145/2699916.

Y. C. Song, H. Kautz, J. Allen, M. Swift, Y. Li, J. Luo, and C. Zhang. A Markov Logic Framework for Recognizing Complex Events from Multimodal Data. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, pages 141–148, New York, NY, USA, 2013a. ACM. ISBN 978-1-4503-2129-7. doi: 10.1145/2522848.2522883. URL http://doi.acm.org/10.1145/2522848.2522883.

Y. C. Song, H. A. Kautz, Y. Li, and J. Luo. A General Framework for Recognizing Complex Events in Markov Logic. In *AAAI Workshop: Statistical Relational Artificial Intelligence*. AAAI, 2013b.

A. Srinivasan. *The Aleph Manual*, 2004. URL http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/. http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/.

A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Comput. Linguist.*, 21(2):165–201, June 1995. ISSN 0891-2017.

B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. MIT Press, 2003.

S. D. Tran and L. S. Davis. Event Modeling and Recognition Using Markov Logic Networks. In *Proceedings of the 10th European Conference on Computer Vision (ECCV)*, volume 5303 of *Lecture Notes in Computer Science*, pages 610–623. Springer, 2008.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, Dec. 2005. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1046920.1088722.

D. L. Vail, M. M. Veloso, and J. D. Lafferty. Conditional random fields for activity recognition. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, page 13311338. IFAAMAS, 2007.

S. Vishwakarma and A. Agrawal. A survey on activity recognition and behavior understanding in video surveillance. *The Visual Computer*, 29(10):983–1009, Oct. 2013. ISSN 0178-2789, 1432-2315.

Y. Wang, X. Li, X. Li, and Y. Wang. A survey of queries over uncertain data. *Knowledge and Information Systems*, 37(3):485–530, Apr. 2013a. ISSN 0219-1377, 0219-3116.

Y. H. Wang, K. Cao, and X. M. Zhang. Complex event processing over distributed probabilistic event streams. *Computers & Mathematics with Applications*, 66(10):1808–1821, Dec. 2013b. ISSN 0898-1221.

S. Wasserkrug, A. Gal, and O. Etzion. A taxonomy and representation of sources of uncertainty in active systems. In O. Etzion, T. Kuflik, and A. Motro, editors, *Next Generation Information Technologies and Systems*, number 4032 in Lecture Notes in Computer Science, pages 174–185. Springer Berlin Heidelberg, Jan. 2006. ISBN 978-3-540-35472-7, 978-3-540-35473-4.

S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Complex event processing over uncertain data. In *Proceedings of the second international conference on Distributed event-based systems*, pages 253–264. ACM, 2008.

S. Wasserkrug, A. Gal, and O. Etzion. A model for reasoning with uncertain rules in event composition systems. *arXiv:1207.1427 [cs]*, July 2012a. URL http://arxiv.org/abs/1207.1427.

S. Wasserkrug, A. Gal, O. Etzion, and Y. Turchin. Efficient processing of uncertain events in rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 24(1):45–58, Jan. 2012b. ISSN 1041-4347. doi: 10.1109/TKDE.2010.204.

E. Wu, Y. Diao, and S. Rizvi. High-performance Complex Event Processing over Streams. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 407–418, New York, NY, USA, 2006. ACM.

T.-y. Wu, C.-c. Lian, and J. Y.-j. Hsu. Joint recognition of multiple concurrent activities using factorial conditional random fields. In *Proceedings of the Workshop on Plan, Activity, and Intent Recognition (PAIR)*, page 8288. AAAI Press, 2007.

J. M. Zelle, C. A. Thompson, M. E. Califf, and R. J. Mooney. Inducing logic programs without explicit negative examples. In *Proceedings of the Fifth International Workshop on Inductive Logic Programming (ILP-95)*, pages 403–416, Leuven, Belgium, 1995. URL http://www.cs.utexas.edu/users/ai-lab/?zelle:ilp95.

H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. *Proc. VLDB Endow.*, 3(1-2):244–255, Sept. 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920875. URL http://dx.doi.org/10.14778/1920841.1920875.

H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. pages 217–228. ACM Press, 2014. ISBN 9781450323765. doi: 10.1145/2588555.2593671.