NATIONAL KAPODISTRIAN UNIVERSITY OF ATHENS

DOCTORAL THESIS

Scalable Relational Learning for Event Recognition

Author:
Nikolaos KATZOURIS

Supervisor:
Prof. Panagiotis RONDOGIANNIS

A thesis submitted in partial fulfilment of the requirements

for the degree of Doctor of Philosophy in the

National Kapodistrian University of Athens

Department of Informatics and Telecommunications

in collaboration with the

National Centre for Scientific Research "Demokritos"

Institute of Informatics and Telecommunications

THESIS COMMITTEE

February 2017

Abstract

Event recognition systems rely on knowledge bases of event definitions to infer occurrences of events in time. Using a logical framework for representing and reasoning about events has several advantages, including robust temporal reasoning via the incorporation of action formalisms, such as the Event Calculus, while it offers direct connections to machine learning, via Inductive Logic Programming (ILP), thus allowing to avoid the tedious and error-prone task of manual knowledge construction. However, learning Event Calculus theories is a challenging task, which most ILP systems cannot fully undertake. A few systems that are capable of rising to the task do not scale to large data volumes, typical of event recognition applications. In this thesis we address the issue of scalable learning for event recognition with ILP and propose two scalable frameworks for the automated construction of event definitions in the form of first-order rules in the Event Calculus. The first of these frameworks is an incremental learner that learns by progressively revising an initial hypothesis in the face of new evidence that arrives over time. It is based on an existing state-of-the-art ILP learner capable of learning theories in the Event Calculus, which however is an "one-shot" system, i.e. it learns whole theories from the entirety of available training examples, resulting in a typically intractable search space. The second of our proposed frameworks adapts a standard hill-climbing ILP search strategy to work in an online fashion, i.e. learning in a single-pass over an arbitrarily large stream of training examples. We present an experimental evaluation for both of our proposed frameworks, using large-scale real and synthetic datasets from the domains of human activity recognition and city transport management.

Acknowledgements

Completing a PhD is, well, hard. It takes devotion and hard work, and demands to get acquainted with the ways of doing research, which is not always easy. One must learn to be patient and thorough, since the road from “an idea that might work” to something that actually does is usually long. Part of the road involves the harder-than-it-seems task of clearly explaining the idea to people who usually know better, which often requires to be able to cope with disappointment productively. The people that surround and support you during the course of a PhD are key to its success. I was fortunate enough to work with some exceptional researchers and colleagues, who I’d like to thank deeply.

I owe a lot to my supervisors, Prof. Panos Rondogiannis from the university of Athens and Dr. Giorgos Paliouras from NCSR “Demokritos”. I had the chance to meet Panos as my teacher during my time at a post-graduate program in logic and computation. I learned a lot from him back then and I was no stranger to his reputation as a researcher. So, I am sincere when I say that I was honoured when he accepted me as his PhD student. He has been there for me ever since, offering his help and support.

I met Giorgos during a period that was not the easiest for me. One of the reasons is that my current job at the time just didn’t seem enough for me. Giorgos offered me the chance to do research and work with his team at NCSR “Demokritos”, while he accepted to co-supervise my PhD, making it all possible financially. That alone is enough for me to be greatly indebted to him. But Giorgos offered many more reasons for that. It’s hard to imagine in what way could I have been better guided during the course of this PhD. Giorgos has been enormously patient in explaining things, listening to poorly conceived arguments, reading bad versions of “first attempts” and so on. At the same time, and while he’s always been involved in a million things, he was also always actively involved in my research, suggesting new ideas and workarounds in a way that cannot stop to impress me. For his knowledge and his will to pass it on, he has my deep admiration and respect.

I had the privilege of having my PhD co-supervised by Dr. Alexander Artikis, who is the head of the complex event recognition group at NCSR “Demokritos”. The completion of this thesis owes a lot to him. In addition to him offering his deep knowledge of artificial intelligence, Alex spent countless hours reading and commenting one poorly-written paper draft after another, and countless more discussing my ideas and suggesting ways to improve them. His persistence in not letting go of attempts that I myself considered as failed is one reason why these attempts finally worked, and a valuable lesson on how to get things done.

During the course of this PhD, I worked at the Software and Knowledge Engineering lab (SKEL), at NCSR “Demokritos”. It’s a vibrant place with lots of people working on interesting stuff, which is motivating in its own right. During my time there, I had the privilege of working in several research projects, which, in addition to getting my PhD funded, has been a great opportunity to get familiarized with interesting ideas and difficult problems. I am deeply grateful to SKEL for giving me this chance and to all the people I worked with. I have some special thanks for my closer colleagues from SKEL's complex event recognition group. Tasos Skarlatidis has been influential in our team, and although he's no longer part of it, he has set some "tech standards". It's hard to be both a good researcher and a good software engineer, and Tasos is, which is somewhat inspiring. Although Elias Alevizos and Vagelis Micheloudakis did not have, strictly speaking, any immediate involvement in the completion of this thesis, they deserve a special credit for making our group worth of wanting to be part of it. Moreover, these guys are really funny.

I am too lucky to have the parents that I do, Ανδρέας and Αλεξάνδρα. I don't think that there is room here (or anywhere, for that matter), for even beginning to explain how much I owe to them. It only seems natural to dedicate this thesis to them (but especially my mom, she has been more consistent in asking "how's that PhD going?")

I am also lucky enough to have great friends. Some of them know from their own experience what's it like to go through a PhD and have offered their advice. All of them have been amazingly supportive in the best way your friends can do that: Being there, along with the beer.

Last but not least, I am deeply grateful to Nefeli, who I met shortly after the beginning of my PhD work, and who has been here since, with endless affection and patience. She's an artistic spirit, she likes to hand-craft things and she's lots of fun. So, she re-assured me early on, that if I proved totally worthless for research, she would make me my own Turing award and even throw a party for it. It's been tempting...

Nikos Katzouris
Athens, February 2017

Contents

List of Figures

List of Tables

1 Introduction

At an increasing rate, information systems need to deal with massive data flows that stream-in from a multitude of sources. Daily activities in business and industry are being automated, probes and sensors are being deployed on infrastructures and physical devices, large data volumes are being produced by virtual agents in enterprise software systems, while an overwhelming amount of information is being exchanged on the web, as various aspects of people's daily lives are moving online. Most of these data are time-stamped, conveying information about *events*. To make sense of these data, the assistance of automated tools is required, and *complex event processing*, as a set of methodologies and techniques for computing with events [Etzion and Niblett, 2010], comes to the rescue.

1.1 Complex Event Recognition

Complex Event recognition, or "*event pattern-matching*" [Luckham, 2001], is a sub-field of complex event processing that seeks to detect interesting event patterns in temporal data. This allows to analyze and extract insights from the data, provide reactive measures in a timely fashion, assist human operators in decision making and so on. Examples include the detection of computer network attacks [Dousson and Le Maigat, 2007], recognition of human activity from videos [Brendel et al., 2011], emerging stories and trends on the web and social networks[1], traffic and transport management [Artikis et al., 2015b], fraud detection in online transactions [Schultz-Møller et al., 2009], medical applications, such as recognition of cardiac arrhythmias [Callens et al., 2008a], epidemic spread [Chaudet, 2006], business process management [Janiesch et al., 2011], maritime surveillance [Patroumpas et al., 2015], assisted living [Storf et al., 2009], the Internet-of-Things [Wang et al., 2013] and so on.

An event is a time-stamped piece of information that represents an occurrence within a system or domain of interest [Etzion and Niblett, 2010]. For instance, an event may be a sensor reading, a video frame, an online activity (e.g. a tweet), a financial transaction, a

[1] https://www.recordedfuture.com/

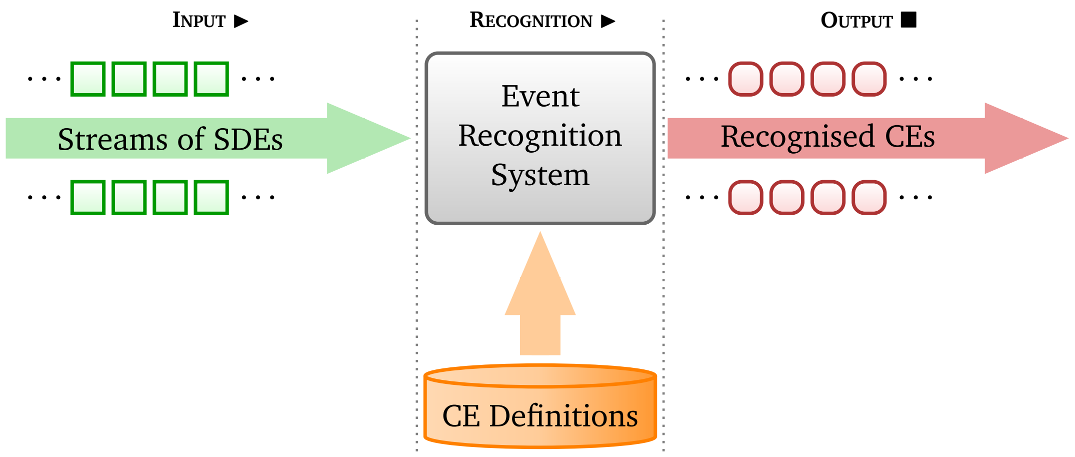

FIGURE 1.1: An illustration of the event recognition process.

GPS signal etc. It may also be a piece of information resulting from some computational process, e.g. a statement representing the fact that a person is walking towards some direction at a particular time, resulting by feeding the person's (x, y) coordinates and her speed over a period of time to an activity classification model; or a statement saying that a credit card transaction is fraudulent, resulting by matching some of the transaction's attributes against a fraud pattern.

An event may be correlated with other events that tend to occur together, or in close temporal proximity. For instance, in a traffic management application, a *traffic congestion* event may be related to a *slow average car speed* event, which in turn results by averaging raw sensor data over a period of time. Such correlations between events may be expressed in rule-like patterns of the form "if a *slow average car speed* event occurs, then a *traffic conjunction* event occurs". An event may occur instantaneously, or it may be durative. For instance, a *fraudulent credit card transaction* event occurs instantaneously, while once a *traffic congestion* event occurs, it persists for a period of time, until some other event that signifies the resolution of the congestion occurs. Additionally, events often involve relations between entities. Consider for instance the event "*person*$_1$ *and person*$_2$ *are walking together at a particular time*" in the context of an activity recognition application, which involves two different entities (person$_1$ and person$_2$).

Figure 1.1 illustrates the event recognition process. The input to an event recognition system consists of a stream of *low-level*, or *simple events*. These are event "primitives", meaning that, in a particular application, their occurrence is assumed not to depend on other events. A knowledge base of event patterns defines *high-level*, or *complex events* of interest, as combinations of simple events, other complex events and potentially additional domain-specific knowledge. At the core of the system lies a reasoning engine that matches the input stream against the event patterns in the knowledge base to recognize complex events. The stream of recognized complex events is passed to the output of the system for further processing.

1.2 Motivation of this Thesis

Event recognition systems may be divided into three broad categories, based on the different processing mechanisms and event pattern specification languages they use Cugola and Margara [2012]. The first category descents from database theory and consists of systems that operate by executing so-called "standing queries", i.e. queries that run constantly and provide updated answers as new data arrives. Event pattern specification languages for this category are extensions of SQL for preforming relational algebra operations on data streams, like selections, aggregates, joins and so on. A well-known, open-source representative of this category is the ESPER[2] system. The second category comprises rule-based systems, where event patterns are sets of rules that fire when their preconditions are satisfied by the incoming events. Most rule-based systems use ad-hoc languages for event pattern specification. Examples of such systems are AMIT [Adi and Etzion, 2004] and DROOLS[3]. Finally, the third category of event recognition systems consists of logical approaches. Systems in this category use a first-order logical formalism to represent event patterns and rely on logical inference to perform event recognition. Some examples of systems in this category are ETALIS [Anicic et al., 2012], RTEC [Artikis et al., 2015b] and SAGE [Broda et al., 2009]. Overviews on systems of all three categories may be found in [Cugola and Margara, 2010, 2012; Paschke, 2006; Paschke and Kozlenkov, 2009].

Logic-based systems have a number of significant advantages over the non-logic-based ones [Artikis et al., 2010a, 2015b, 2012; Paschke, 2006; Paschke and Kozlenkov, 2009]. In many real-life applications, temporal reasoning and event recognition require modeling the effects of event occurrences on several properties of a time-evolving system, as well as the duration of such effects. This is a well-studied problem in the field of artificial intelligence and several temporal logical formalisms exist, designed precisely for that task, such as the Event Calculus [Kowalski and Sergot, 1986b]. Such formalisms may be easily incorporated in logic-based event recognition systems, in contrast to non-logic-based systems.

Additionally, logic-based systems exhibit a formal, declarative semantics. In contrast, non-logic based event recognition systems typically have an informal, procedural semantics [Cugola and Margara, 2010; Eckert and Bry, 2010; Paschke, 2006; Paschke and Kozlenkov, 2009]. As pointed-out in [Paschke, 2006; Paschke and Kozlenkov, 2009], this is a serious omission in many real-life applications, where it is crucial to be able to trace and validate several aspects of the event recognition process, such as the effects of occurring events. Moreover, logic-based event recognition systems allow to represent and reason with complex relations between entities and utilize rich background knowledge, contrary to non-logic-based systems.

[2]http://www.espertech.com/esper/
[3]http://www.drools.org/

An additional important advantage of logic-based systems is that they offer direct connections to *machine learning*. Typically in an event recognition system, the event specification patterns are manually authored by human domain experts. This is an expensive, time consuming and error prone task [Artikis et al., 2010a], and a limiting factor for the diffusion of event recognition systems [Margara et al., 2014]. To alleviate the problem, machine learning techniques may be used to automatically construct event patterns from data. The field of Inductive Logic Programming (ILP) [De Raedt, 2008; Lavrač and Džeroski, 1993; Muggleton and De Raedt, 1994; Nienhuys-Cheng and De Wolf, 1997] lies at the intersection of machine learning and logic programming and provides tools and algorithms for learning logical theories from relational data. With ILP, expressive relational event specification patterns in the form of first-order rules may be generated in an automated way and used directly for reasoning in a logic-based event recognition system.

In contrast, learning event patterns from data in non-logic based event recognition approaches is still hard, due to the fact that such approaches use diverse and ad-hoc reasoning tools and event pattern specification languages. A few approaches have been proposed [Margara et al., 2014, 2013], but they themselves resort to ad-hoc machine learning algorithms. Such algorithms are hard to evaluate outside their domain of use.

The Event Calculus has been used for event recognition in the past [Cervesato and Montanari, 2000; Chittaro and Dojat, 1997; Paschke, 2006; Paschke and Bichler, 2008; Paschke et al., 2010]. A downside of such approaches is that they are considered less efficient than the non-logic-based ones [Artikis et al., 2015b]. This is the case with most logic-based event recognition systems, but not with RTEC (Run Time Event Calculus) [Artikis et al., 2015b], a recent Event Calculus-based system for event recognition. RTEC uses reasoning over time intervals and various optimization techniques for efficient event recognition and is able to scale to large data volumes. Moreover, it has been evaluated in a number of challenging large-scale applications [Artikis et al., 2015b, 2014; Patroumpas et al., 2016, 2015].

However, learning Event Calculus theories with ILP remains a challenging task [Katzouris et al., 2015; Ray, 2009a] that only very few ILP systems can fully undertake [Corapi et al., 2012; Ray, 2009a]. These systems though, are only able to learn from small datasets and do not scale to the volumes of data collected in event recognition applications.

This is the problem that we address in this thesis: The development of scalable algorithms for learning domain-specific event patterns in the form of Event Calculus theories, from large volumes of temporal data. Throughout the thesis we use machinery based on logic programming, where rules (*clauses*, in a logic programming context) are definitions for first-order predicates. Therefore, we henceforth refer to event specification patterns as *event definitions*.

1.3 Thesis Contribution

In the remainder of this thesis, we present two scalable machine learning approaches for the automated construction of event definitions in the form of Event Calculus theories. Moreover we present experimental evaluation of various aspects of these approaches on real and synthetic datasets. The specific contribution of the thesis is summarized below.

1.3.1 Incremental Learning of Event Definitions

Learning Event Calculus theories in the form of logic programs imposes two main challenges: (a) it requires learning in a non-monotonic setting, due to the Negation as Failure operator [Clark, 1977] that it uses as a means for modeling *inertia*, i.e. persistence of properties of the world through time [Mueller, 2014]; and (b) it requires to derive possible causes of observed events at learning time [Moyle and Muggleton, 1997]. Traditional Inductive Logic Programming systems cannot rise to these challenges, since they are either restricted to Horn logic, or they lack a robust Negation as Failure semantics [Ray, 2009a; Sakama, 2000], and their abilities to reason with missing, or indirectly observable knowledge are limited [Muggleton, 1995a].

Non-monotonic Inductive Logic Programming offers a solution to both (a) and (b) above, by utilizing Abductive Logic Programming (ALP) [Denecker and Kakas, 2002; Kakas et al., 1993; Kakas and Mancarella, 1990]. Abduction in logic programming is given a non-monotonic semantics [Eshghi and Kowalski, 1989] and in addition, it is by nature an appropriate framework for reasoning with incomplete knowledge. A number of such non-monotonic Abductive-Inductive Logic Programming systems exists, that are able to learn theories in the Event Calculus [Corapi et al., 2010; Ray, 2009a].

However, in a non-monotonic setting, a theory cannot be learnt one clause at a time, as is common in traditional Inductive Logic Programming approaches, since adding new clauses to a theory may invalidate previously constructed ones. As a result, the aforementioned non-monotonic learners are forced to learn the entire theory in one shot, using all training examples simultaneously. This results in an intractable search space, even with relatively small amounts of data.

In this thesis we address this issue, by building upon one such non-monotonic learner, XHAIL[Ray, 2009a], and proposing a methodology for scaling-up its core functionality to large data volumes, via an *incremental learning* approach. In more detail, our proposed system, ILED (Incremental Learning of Event Definitions) [Katzouris et al., 2015] has the following features:

- It is designed to work with training examples that arrive over time, by revising previously constructed hypotheses to fit new observations. This is a particularly suitable setting for event-oriented learning tasks, where data are often collected at different times and under various circumstances, or they arrive in streams. On such occasions, batch learning systems have no alternative but discarding past hypotheses and learning from scratch, which is a bottleneck for scalability, especially in cases where the data arrive with high velocity.

- ILED revises a hypothesis so that it accounts for the entirety of accumulated experience, in addition to incoming observations. Moreover, it does so with a *single pass* over the past data, while ensuring the soundness of the outcome.
- In the case where the entire dataset is in place when learning begins, ILED scales-up the XHAIL methodology by splitting the dataset into data chunks and learning from each chunk separately, in a sequential fashion, thus "simulating" the incremental setting described above. Given a dataset D of size k, XHAIL would learn a hypothesis in one shot, form the entirety of k examples. The search space for this task is typically huge, due to increased combinatorial complexity, even for small values of k. In contrast, ILED splits D into n data chunks, each consisting of m training examples, and learns a sound hypothesis by operating on it at most twice, therefore, with no more than $2n$ operations in the worst case. Moreover, the unit cost of operating on the revisable hypothesis w.r.t. a single chunk depends on the chunk size m, and it may be kept low, by partitioning D into sufficiently small data chunks. Therefore, ILED may scale to datasets of arbitrary size, while ensuring the soundness of the learnt theory. The price to pay for this scalability is that ILED learns slightly larger, more complex theories than XHAIL.

1.3.2 Online Learning of Event Definitions

Learning sound hypotheses, as ILED does, is desirable, however it is often of low practical value. Most of the time, real-life data contain noise and the induction of sound hypotheses from such data is not possible. In such cases it is preffered to relax the requirement for soundness and learn a less-than-perfect model with an adequate fit in the data, rather than nothing at all. Additionally, preserving a full memory of past experience and requiring "backward compatibility" from a model that is constantly updated in the face of new evidence, as ILED does, is also often impractical. In many event-related applications, data arrive at a high velocity, in continuous, potentially infinite streams, which are impossible to store for offline analysis. Methods that extract insights from such streams need to be capable of fast processing of new data and building a decision model by a single pass over the training data [Gama, 2010; Gama and Gaber, 2007].

To address these issues, we present OLED (Online Learning of Event Definitions) [Katzouris et al., 2016], a method that learns event definitions in the form of Event Calculus theories in an online fashion, with a single pass over a data stream. To do so, we utilize a methodology widely used in stream mining, based on so-called (ϵ, δ)-approximations

[Gaber et al., 2014]. Learning algorithms based on this methodology use statistical tools to make decisions that are optimal within an error margin ϵ and with some probability $1-\delta$, where δ is a user-defined parameter. This allows for making decisions using limited resources, in our case, a limited amount of data from the training stream. In more detail, OLED has the following features:

- OLED is capable of handling noisy data by learning imperfect theories that account for most of the positive examples, as well as some of the negatives.
- It adapts a standard Inductive Logic Programming clause learning technique based on hill-climbing, to work in an online fashion. OLED learns gradually specializes an over-general clause by evaluating its candidate specializations, much like the well-known FOIL [Quinlan, 1990a] Inductive Logic Programming system does. However, instead of evaluating the candidate specializations on the entire training set, which is practically infeasible, OLED relies on the *Hoeffding bound* [Hoeffding, 1963] to find specializations that are only (ϵ, δ)-optimal, using a limited amount of training data.
- Using this online hill-climbing technique requires to learn each clause independently. This is hard to do in the case of Event Calculus theories, because candidate clauses depend on each other via the core axioms of the Event Calculus. To address this issue, OLED uses a technique that allows to evaluate clauses in isolation, based on a scoring function that takes into account the potential utility of each clause in a theory that results by joining these clauses together.

1.3.3 Publications

Parts of this thesis have been published in the following papers.

Journal publications:

- Katzouris N., Artikis, A. and Paliouras, G. (2016) “Online learning of event definitions”, *Theory and Practice of Logic Programming*, 16(5-6), pp. 817-833.
- Katzouris N., Artikis A., Paliouras G. (2015) “Incremental Learning of Event Definitions with Inductive Logic Programming”. *Machine Learning*, 100(2-3), pp. 555-585.

Conference publications:

- Patroumpas K., Artikis A., Katzouris N., Vodas M., Theodoridis Y., and Pelekis N. (2015) “Event Recognition for Maritime Surveillance”, *International Conference on Extending Database Technology (EDBT)* pp. 629-640.

- Billis A., Katzouris N., Artikis A. and Bamidis P. (2015), “Clinical Decision Support for Active and Healthy Ageing: an intelligent monitoring approach of daily living activities”. *Portuguese Conference on Artificial Intelligence*, pp. 128-133.

- Katzouris N., Artikis A. and Paliouras G. (2014) “Event Recognition for Unobtrusive Assisted Living”, *Hellenic Conference on Artificial Intelligence*, pp. 475-488.

Workshop publications:

- Katzouris N., Artikis A. and Paliouras G. (2015) “Semi-Supervised Learning of Event Calculus Theories”, *Proceedings of the ECML-2015 Doctoral Consortium*.

1.4 Thesis Outline

The remainder of this thesis is structured as follows: In chapter 2 we present the basics of the Event Calculus and Inductive Logic Programming. We also discuss in detail the difficulties of learning Event Calculus theories with Inductive Logic Programming and present XHAIL, as well as a number of related abductive-inductive learners. We also discuss related work on learning from temporal data with Inductive Logic Programming. In Chapter 3 we present the basics of theory revision in Inductive Logic Programming and we then discuss ILED in detail, while in Chapter 4 we present an experimental evaluation for ILED. In Chapter 5 we present the OLED system, after some basic background on learning from data streams, while in Chapter 6 we present OLED’s experimental evaluation. Finally, in Chapter 7 we indicate directions for future work and conclude.

2 Background

This chapter provides some necessary background material for the thesis. We begin with basic notions from logic programming in the subsequent paragraphs. We then review the Event Calculus formalism and the fundamentals of Inductive Logic Programming, using a running example of the learning problem we address, which is used frequently in the remainder of the thesis. The chapter concludes with a review of related work.

2.1 Logic Programming Basics

In what follows, we assume a first-order language where constants, variables, predicate/-function symbols and terms are defined in the regular way [Lloyd, 1987]. Following Prolog's convention, throughout this thesis predicates and ground (variable-free) terms in logical formulae start with a lower case letter, while variable terms start with a capital letter. An atom is an expression of the form $p(t_1, \dots, t_n)$, where $t_1, \dots, t_n$ are terms, and a literal is either an atom or a negated atom, i.e. an expression of the form $\mathsf{not}\ p(t_1, \dots, t_n)$, where not denotes Negation as Failure [Clark, 1978]. A clause is an expression of the form $\alpha \leftarrow \delta_1, \dots, \delta_n$, where α is an atom, called the *head* of the clause and $\delta_1, \dots, \delta_n$ are literals, which collectively form the *body* of the clause. Commas in clause bodies denote conjunction, therefore the expression $\alpha \leftarrow \delta_1, \dots, \delta_n$ is equivalent to $\alpha \leftarrow \delta_1 \wedge \dots \wedge \delta_n$, and when readability requires it we use the latter notation. We use lower-case Greek letters to denote clause literals and lower-case Latin letters to refer to clauses, as in "let r be the clause $\alpha \leftarrow \delta_1, \dots, \delta_n$".

A *normal logic program* is a collection of clauses. A *Horn logic program* is a collection of *definite clauses*, i.e. clauses whose body contains non-negated literals only. The terms "logic program" and "theory" are used interchangeably in what follows. A *unit clause* is a clause with an empty body, denoted by $\alpha \leftarrow$, which is shorthand for $\alpha \leftarrow \mathsf{true}$. An *integrity constraint* is a clause with an empty head, denoted by $\leftarrow \delta_1, \dots, \delta_n$, which is shorthand for $\mathsf{false} \leftarrow \delta_1, \dots, \delta_n$. We use capital-case Latin letters to denote logic programs.

Although different semantics is possible, in this work we assume that all logic programs are subject to the stable model semantics for logic programs with Negation as Failure [Baral, 2003; Gebser et al., 2012; Gelfond, 2008; Gelfond and Lifschitz, 1988]. Given a logic program P, a Herbrand interpretation I is a subset of the set of all possible groundings of P. I satisfies a literal a (resp. not a) iff $a \in I$ (resp. $a \notin I$). I satisfies a set of ground atoms iff it satisfies each one of them and it satisfies a ground clause iff it satisfies the head, or does not satisfy at least one body literal. I is a *Herbrand model* of P iff it satisfies every ground instance of every clause in P and it is a *minimal Herbrand model* iff no strict subset of I is a model of P. I is a *stable model* of P iff it is a minimal Herbrand model of the Horn program that results from the ground instances of P after the removal of all clauses with a negated literal not satisfied by I, and all negative literals from the remaining clauses. Each Horn logic program has a unique minimal Herbrand model, which coincides with its stable model. In contrast, a normal logic program may have no, or more that one stable models. There are two primal alternatives for the logical entailment relation under the stable model semantics. In the *cautious* (resp. *brave*) version, a program P_1 logically entails a program P_2 (den. $P_1 \vDash P_2$) iff *every* (resp. *at least one*) stable model of P_1 satisfies P_2. Unless otherwise stated, in this thesis the cautious version of entailment is adopted.

2.2 The Event Calculus

The Event Calculus [Kowalski and Sergot, 1986a] is a first-order temporal logic for reasoning about events and their effects, which has been successfully used in numerous event recognition applications [Artikis et al., 2015a; Cervesato and Montanari, 2000; Chaudet, 2006; Katzouris et al., 2014; Paschke, 2005; Patroumpas et al., 2015]. Several dialects of the Event Calculus have been proposed over the years, either in logic programming or in full first-order logic – see [Miller and Shanahan, 2002; Mueller, 2008, 2014] for a review. Most of these dialects share an ontology consisting of *time points*, *fluents* and *events*, along with a set of *domain-independent axioms*. Time points are simply integers or real numbers; Fluents are properties which have certain values in time; and events are occurrences in time that may affect fluents and alter their value. The set of domain-independent axioms model the common sense *law of inertia*, according to which fluents persist over time, unless they are affected by the occurrence of an event. Typically, an Event Calculus program contains additionally a set of *domain-specific axioms*, which specify how events effect fluents. We next discuss the domain independent/specific axiomatization of the Event Calculus in further detail.

Predicate	**Meaning**
$\mathsf{happensAt}(E, T)$	Event E occurs at time T
$\mathsf{initiatedAt}(F, T)$	At time T a period of time for which fluent F holds is initiated
$\mathsf{terminatedAt}(F, T)$	At time T a period of time for which fluent F holds is terminated
$\mathsf{holdsAt}(F, T)$	Fluent F holds at time T
Axioms	
$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\mathsf{initiatedAt}(F, T).$	$\mathsf{holdsAt}(F, T+1) \leftarrow$ $\mathsf{holdsAt}(F, T),$ $\mathsf{not\ terminatedAt}(F, T).$

TABLE 2.1: The basic predicates and domain-independent axioms of the SDEC dialect.

2.2.1 Domain-independent Axioms

We assume a linear time model, where time points range over the set of integers. The complete formulation of the Event Calculus may be found in [Mueller, 2008, 2014], but its presentation is beyond the scope of this thesis, we thus omit it. Instead, we focus on the particular dialect that we employ in this work, whose expressive power suffices for most event recognition tasks [Artikis et al., 2015a]. This dialect is a simplified version of the Discrete Event Calculus DEC[1]. It can be shown that the DEC dialect is logically equivalent to the complete Event Calculus when time ranges over the set of integers [Mueller, 2008].

The building blocks of our dialect, which we call SDEC (Simplified Discrete Event Calculus) and its domain-independent axioms are presented in Table 2.1. The first axiom in Table 2.1 states that a fluent F holds at time T if it has been initiated at the previous time point, while the second axiom states that F continues to hold unless it is terminated.

The basic difference between SDEC and DEC is that the latter contains some additional axioms that involve four more predicates: releases and releasedAt, which are domain-specific predicates and are used to specify the conditions under which the law of inertia for a fluent is disabled; and the trajectory and antiTrajectory predicates, that are used to model the change of a fluent, based on the effects of some domain-specific function, e.g. the height of a falling ball [Mueller, 2014]. We omit the axioms that define the releases, releasedAt, trajectory and antiTrajectory predicates and refer to [Mueller, 2014] for further details. By removing these axioms from DEC (thus obtaining the SDEC dialect), we assume that all fluents are always subject to inertia and we do not allow

[1]An open-source implementation of the Discrete Event Calculus is available in `http://decreasoner.sourceforge.net`

Narrative	**Annotation**
......	......
$\mathsf{happensAt}(inactive(id_1), 999)$ $\mathsf{happensAt}(active(id_2), 999)$ $\mathsf{holdsAt}(coords(id_1, 201, 432), 999)$ $\mathsf{holdsAt}(coords(id_2, 230, 460), 999)$ $\mathsf{holdsAt}(direction(id_1, 270), 999)$ $\mathsf{holdsAt}(direction(id_2, 270), 999)$	$\mathsf{not\ holdsAt}(moving(id_1, id_2), 999)$
$\mathsf{happensAt}(walking(id_1), 1000)$ $\mathsf{happensAt}(walking(id_2), 1000)$ $\mathsf{holdsAt}(coords(id_1, 201, 454), 1000)$ $\mathsf{holdsAt}(coords(id_2, 230, 440), 1000)$ $\mathsf{holdsAt}(direction(id_1, 270), 1000)$ $\mathsf{holdsAt}(direction(id_2, 270), 1000)$	$\mathsf{not\ holdsAt}(moving(id_1, id_2), 1000)$
$\mathsf{happensAt}(walking(id_1), 1001)$ $\mathsf{happensAt}(walking(id_2), 1001)$ $\mathsf{holdsAt}(coords(id_1, 201, 454), 1001)$ $\mathsf{holdsAt}(coords(id_2, 227, 440), 1001)$ $\mathsf{holdsAt}(direction(id_1, 275), 1001)$ $\mathsf{holdsAt}(direction(id_2, 278), 1001)$	$\mathsf{holdsAt}(moving(id_1, id_2), 1001)$
......	......

TABLE 2.2: An annotated stream of low-level events

representing and reasoning about discrete change of fluents, under the effects of domain-specific functions. These simplifications improve significantly the efficiency of the SDEC reasoning engine, while they do not compromise the expressive power of the SDEC formalism, as far as event recognition applications are concerned [Artikis et al., 2015a]. In what follows and unless otherwise stated, by "Event Calculus" we refer to the SDEC dialect.

2.2.2 Domain-specific Axioms: An activity recognition use-case

initiatedAt/2 and terminatedAt/2 predicates in the axioms of Table 2.1 are defined in an application-specific manner by a set of *domain-specific* axioms that specify the way in which the occurrence of events affects fluents by changing their truth value. To illustrate the domain-specific axiomatization of the Event Calculus, we use the task of activity recognition, as defined in the CAVIAR[2] project. The same task is used frequently as a running example in the remainder of this thesis.

The CAVIAR dataset consists of videos of a public space, where actors walk around, meet each other, browse information displays, fight and so on. These videos have been manually annotated by the CAVIAR team to provide the ground truth for two types of activity. The first type corresponds to low-level events, that is, knowledge about a person's activities at a certain time point (for instance *walking*, *running*, *standing still* and

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

initiatedAt$(moving(X,Y),T) \leftarrow$	terminatedAt$(moving(X,Y),T) \leftarrow$
happensAt$(walking(X), T)$,	happensAt$(inactive(X), T)$,
happensAt$(walking(Y), T)$,	not holdsAt$(close(X, Y, 30), T)$.
holdsAt$(close(X, Y, 25), T)$,	
holdsAt$(orientation(X, Y, 45), T)$	

TABLE 2.3: Two domain-specific axioms for the domain of activity recognition

so on). The second type corresponds to high-level events, activities that involve more than one person, for instance two people *moving together*, *fighting*, *meeting* and so on. The aim is to recognize high-level events by means of combinations of low-level events and some additional domain knowledge, such as a person's position and direction at a certain time point.

Low-level events are represented in SDEC by streams of ground happensAt/2 atoms (see Table 2.2), while high-level events and other domain knowledge are represented by ground holdsAt/2 atoms. Streams of low-level events together with domain-specific knowledge will henceforth constitute the *narrative*, while knowledge about high-level events is the *annotation*. Table 2.2 presents an annotated stream of low-level events. We can see for instance that the person id_1 is $inactive$ at time 999, her (x, y) coordinates are $(201, 432)$ and her direction is $270°$. The annotation for the same time point informs us that id_1 and id_2 are not moving together. Fluents are used to express both high-level events and narrative knowledge, such as the coordinates of a person. We discriminate between *inertial* and *statically defined* fluents. High-level events are of the former type, and they should be inferred by the Event Calculus axioms, while parts of the narrative expressed by means of fluents are of the latter type and are provided with the input. For instance holdsAt$(moving(id_1, id_2), 100)$ is an inertial fluent (since it represents a high-level event), while holdsAt$(close(id_1, id_2, 25), 100)$ is a statically defined fluent, since it is part of the narrative.

A set of domain-specific axioms in the Event Calculus is a set of clauses that define conditions under which fluents are initiated or terminated. As an example, consider the two clauses in Table 2.3. The first clause dictates that *moving* of two persons X and Y is initiated at time T if both X and Y are walking at time T, their euclidean distance is less than 25 and their difference in direction is less than $45°$. The second clause dictates that *moving* of X and Y is terminated at time T if one of them is standing still at time T (exhibits an inactive behavior) and their euclidean distance at T is greater that 30.

A set of domain-specific axioms, i.e. definitions for initiatedAt/2 and terminatedAt/2 predicates, specify how the occurrence of simple events affects the truth values of fluents that represent complex events, by initiating or terminating them. In what follows we use the terms "domain-specific axioms" and "(complex) event definitions" interchangeably.

2.2.3 Other Action Formalisms

In addition to the Event Calculus, various *action formalisms*, i.e. formal frameworks for reasoning about time and change have been proposed within the field of artificial intelligence. Prominent examples include the Situation Calculus [McCarthy, 2002], the Fluent Calculus [Thielscher, 1999], action language $\mathcal{C}+$ [Akman et al., 2004; Giunchiglia et al., 2004] and Temporal Action Logics [Doherty et al., 1998].

The ontologies of all action formalisms rely on two basic sorts of entities: Events (or actions) Kowalski and Sergot [1986b]; McCarthy [2002], which are instantaneous occurrences; and fluents (or features) [Sandewall, 1992], properties that are used to describe the state of the world and are inert, i.e. they persist in time unless they are affected by an event. An important difference between action formalisms however, is the representation of time. The Situation Calculus and the Fluent Calculus allow for different possible futures by using a multiple time-line model, where each point in time has a single predecessor point, but it may have multiple successor points. Also events are atemporal and sequential: A point in time is a *situation*, the result of a particular sequence of events. In contrast, the Event Calculus, $\mathcal{C}+$ and Temporal Action Logics, assume a single time-line on which events occur. For the purposes of event recognition, a single time-line model is preferred, since it allows to express temporal constraints between events.

2.3 Inductive Logic Programming

Given a domain description in the language of SDEC, the aim of machine learning addressed in this thesis is to derive as set of domain-specific axioms that provide definitions for the complex events in the domain, in terms of their initiation and termination conditions. To this end, we use Inductive Logic Programming (ILP) [De Raedt, 2008; Lavrač and Džeroski, 1993; Muggleton and De Raedt, 1994; Nienhuys-Cheng and De Wolf, 1997], a sub-field of machine learning, where the goal is to induce definitions for some target predicates, given positive and negative examples of these predicates. In this section, we present the basics of ILP and outline the main challenges involved in using ILP for learning theories in the Event Calculus.

2.3.1 The Learning Setting

An ILP task is a triplet $ILP\langle B, E, M\rangle$ where B is a normal logic program called background knowledge, M is some language bias, i.e. a set of directives of some form, that specify the structure of target clauses and $E = E^{+} \cup E^{-}$ is a set of positive (E^{+}) and negative (E^{-}) examples for the target predicates, i.e. the predicates whose definitions we wish to learn. Typically in ILP, positive examples are assumed to be encoded by a set of logical facts (ground unit clauses), while negative examples are either given explicitly by a different set of logical facts, or they are obtained implicitly from the positive examples, by assuming a closed world (i.e. any instance of a target predicate that is not contained in the positive examples is considered a negative one).

Given an ILP task $ILP\langle B, E, M\rangle$, the goal is to derive an *inductive hypothesis* H, i.e. a non-ground normal logic program H in the language defined by M, such that $B \cup H$ *covers* (den. $cov(B \cup H, E^{+})$) the positive and *does not cover* (den. $\neg\, cov(B \cup H, E^{-})$) the negative examples. The coverage relation is typically defined in two alternative ways based on the learning setting, i.e. whether we are *Learning from Entailment* [De Raedt, 2008; Muggleton and De Raedt, 1994; Plotkin, 1970], or *Learning from Interpretations* [De Raedt, 1997, 2008; De Raedt and Džeroski, 1994]. We next review these learning settings and fix a definition for the coverage relation for the remainder of this thesis.

In the Learning from Entailment setting, a training example e is typically a single logical atom and the coverage relation is defined via logical entailment, i.e. $covers(B \cup H, e)$ iff $B \cup H \vDash e$. In the Learning from Interpretations setting, a training example is a Herbrand interpretation, i.e. a set of logical facts. We then say that a hypothesis covers the example e (relative to the background knowledge) iff e is a model of $B \cup H$, i.e. $e \vdash B \cup H$. In particular, in the case of Horn logic programs, $covers(B \cup H, e)$ iff e is the minimal Herbrand model of $B \cup H$; in the case of normal logic programs $covers(B \cup H, e)$ iff the program $B \cup H \cup e$ is satisfiable, i.e. it has at least one stable model.

The selection of a particular learning setting depends on the learning problem at hand [De Raedt, 2008]. Each setting has its own merits. Specifically, the Learning from Interpretations setting is considered more scalable than the Learning from Entailment setting, since each training interpretation is assumed to be independent from the others, in essence, a "small database that may be queried independently" [Blockeel et al., 1999]. In contrast, the main source of inefficiency when learning from entailment lies in the assumption that several examples may be related to each other, so they cannot be handled independently. The prize to pay for the scalability of Learning from Interpretations is that it is less general than its counterpart, i.e. the class of hypotheses learnable from interpretations is a strict subset of those learnable from entailment (e.g. the former cannot learn recursive definitions, while that latter can). However, as pointed out in [Blockeel et al., 1999], Learning from Interpretations suffices for most learning tasks in ILP. We therefore adopt this more scalable framework in the remainder of this thesis and fix the coverage relation in what follows to $cov(B \cup H, e)$ iff $B \cup e$ is a stable model of H.

In our learning setting, a training interpretation is a set of ground atoms consisting of anything known true at *at least* two consecutive time points T and $T{+}1$. For instance, if we denote by e_{999}, e_{1000} and e_{1001} the knowledge (narrative and annotation) about time points $999, 1000$ and 1001 in Table 2.2, each of the following sets of atoms is a training interpretation:

$$I_1 = e_{999} \cup e_{1000}, I_2 = e_{1000} \cup e_{1001}, I_3 = e_{999} \cup e_{1000} \cup e_{1001}$$

This formulation of training examples is possible, due to the domain-independent axioms of SDEC (see Table 2.1), which allow the initiation/termination of complex events to depend only on the narrative and annotation of the previous time-point. Therefore, each interpretation containing knowledge about at least two consecutive time points is independent from others and may be queried in isolation. This complies with the requirements of the Learning from Interpretations setting and it allows for more scalable learning.

Finally, throughout this thesis, and unless otherwise stated, we assume that negated annotation instances are obtained via the closed world assumption. To avoid confusion however, we show negated annotation instances in tables and examples throughout the text (e.g. see Table 2.2).

2.3.2 The Hypothesis Space

ILP algorithms search a hypothesis space to find a logical theory that matches some quality criteria. This space is defined by means of some language bias (the M in the ILP task $ILP\langle B, E, M\rangle$) and is typically structured by generality. A program H_1 is at least as general as a program H_2, iff all the examples covered by H_2 are also covered by H_1. We begin the description of the hypothesis space by the presentation of *mode declarations* [Muggleton, 1995a], a widely used language bias in ILP, and the definition of the hypothesis language. We then discuss structuring the hypothesis space by means of θ-subsumption, the formalization of generality in logic programming.

Mode declarations. A *mode declaration* is an atom of the form modeh(s) or modeb(s), where s is called a schema. A schema s is a ground literal containing *placemarkers*. A placemarker is one of $+$type (input placemarker), $-$type (output placemarker) or $\#$type (ground placemarker), where type is a constant. mode declarations serve as templates for constructing literals, by replacing placemarkers by actual variables or constants, which are "typed", i.e. they are assumed to be of the "sort" indicated by the type of the corresponding placemarker. These literals constructed out of mode declaration atoms, may subsequently be used to construct hypothesis clauses. In particular, a set M of mode declarations defines a language $\mathcal{L}(M)$, as follows: A clause C is in $\mathcal{L}(M)$ iff its head atom (respectively each of its body literals) is constructed from the schema s in a modeh(s) atom (resp. in a modeb(s) atom) in M by: (a) replacing an output placemarker by a new variable; (b) replacing an input placemarker by a variable that appears in the head atom, or in a previous body literal; (c) replacing a ground placemarker by a ground term. A hypothesis H is in $\mathcal{L}(M)$ iff $C \in \mathcal{L}(M)$ for each $C \in H$.

1. $\mathsf{modeh}(\mathsf{initiatedAt}(moving(+\mathsf{person}, +\mathsf{person}), +\mathsf{time}))$
2. $\mathsf{modeh}(\mathsf{terminatedAt}(moving(+\mathsf{person}, +\mathsf{person}), +\mathsf{time}))$
3. $\mathsf{modeb}(\mathsf{happensAt}(walking(+\mathsf{person}), +\mathsf{time}))$
4. $\mathsf{modeb}(\mathsf{not\ happensAt}(walking(+\mathsf{person}), +\mathsf{time}))$
5. $\mathsf{modeb}(\mathsf{happensAt}(active(+\mathsf{person}), +\mathsf{time}))$
6. $\mathsf{modeb}(\mathsf{not\ happensAt}(active(+\mathsf{person}), +\mathsf{time}))$
7. $\mathsf{modeb}(\mathsf{happensAt}(inactive(+\mathsf{person}), +\mathsf{time}))$
8. $\mathsf{modeb}(\mathsf{not\ happensAt}(inactive(+\mathsf{person}), +\mathsf{time}))$
9. $\mathsf{modeb}(\mathsf{happensAt}(running(+\mathsf{person}), +\mathsf{time}))$
10. $\mathsf{modeb}(\mathsf{not\ happensAt}(running(+\mathsf{person}), +\mathsf{time}))$
11. $\mathsf{modeb}(\mathsf{happensAt}(abrupt(+\mathsf{person}), +\mathsf{time}))$
12. $\mathsf{modeb}(\mathsf{not\ happensAt}(abrupt(+\mathsf{person}), +\mathsf{time}))$
13. $\mathsf{modeb}(\mathsf{holdsAt}(close(+\mathsf{person}, +\mathsf{person}, \#\mathsf{distance}), +\mathsf{time}))$
14. $\mathsf{modeb}(\mathsf{not\ holdsAt}(close(+\mathsf{person}, +\mathsf{person}, \#\mathsf{distance}), +\mathsf{time}))$
15. $\mathsf{modeb}(\mathsf{holdsAt}(orientation(+\mathsf{person}, +\mathsf{person}, \#\mathsf{direction}), +\mathsf{time}))$
16. $\mathsf{modeb}(\mathsf{not\ holdsAt}(orientation(+\mathsf{person}, +\mathsf{person}, \#\mathsf{direction}), +\mathsf{time}))$

TABLE 2.4: A set of mode declarations for the domain of activity recognition.

Example 2.1. *Table 2.4 presents a set of mode declarations for the domain of activity recognition. For instance, declarations 1 & 2 dictate that heads of target clauses may be atoms of the form* $\mathsf{initiatedAt}(moving(X, Y), T)$ *or* $\mathsf{terminatedAt}(moving(X, Y), T)$*, where* X, Y *are input variables of type* $+\mathsf{person}$ *and* T *is an input variable of type* $+\mathsf{time}$*. Mode declarations 3-16 specify the form of possible body literals in the target clauses. E.g. modes 3 & 4 state that a body literal may be of the form* $\mathsf{happensAt}(walking(X), T)$*, or not* $\mathsf{happensAt}(walking(X), T)$*, where variables* X *and* T *are input variables of type* $+\mathsf{person}$ *and* $+\mathsf{time}$ *respectively. Similarly, Mode 13 dictates that* $\mathsf{holdsAt}(close(X, Y, 35), T)$ *is also a valid body literal in a target hypothesis clause (note the "#" in front of the distance placemarker in mode 13, indicating that a term that replaces this placemarker should be a constant. Output variable placemarkers (not present in any of the mode declarations in Table 2.4), allow free variables to appear in a clause. For instance, the clause* $p(X, Y) \leftarrow q(X, Z), r(Z, Y)$ *belongs to the language defined by the following set of mode declarations:* $M = \{\ \mathsf{modeh}(p(+\mathsf{any}, +\mathsf{any})), \mathsf{modeb}(q(+\mathsf{any}, -\mathsf{any})), \mathsf{modeb}(r(-\mathsf{any}, +\mathsf{any}))\ \}$

θ-subsumption. A clause r_1 θ-subsumes a clause r_2, denoted $r_1 \preceq r_2$, if there exists a substitution θ such that $head(r_1)\theta = head(r_2)$ and $body(r_1)\theta \subseteq body(r_2)$. Two clauses r_1 and r_2 are equivalent under θ-subsumption if $r_1 \preceq r_2$ and $r_2 \preceq r_1$. A logic program program Π_1 θ-subsumes a program Π_2 if for each clause $r_1 \in \Pi_1$ there exists a clause $r_2 \in \Pi_2$ such that $r_1 \preceq r_2$. A clause r_1 (resp. program Π_1) is more general than a clause r_2 (resp. program Π_2) if $r_1 \preceq r_2$ (resp. $\Pi_1 \preceq \Pi_2$), in which case r_2 (resp. Π_2) is more specific than r_1 (resp. Π_1).

Given a clause r, the set of clauses that θ-subsume r and the set of clauses that are θ-subsumed by r form a lattice [G.Plotkin, 1970]. Generalizing a clause r, or "moving-up" into the θ-subsumption lattice, means to replace r with a clause r' such that $r' \preceq r$. A generalization operator is a function that generates such an r', by e.g. removing a literal from r or replacing a constant with a variable. In contrast, to specialize r, or "moving-down" into the θ-subsumption lattice, means to replace it with r', such that $r \preceq r'$. A specialization operator does this by e.g. adding a literal to the body of r, or replacing a variable with a constant.

Most ILP algorithms construct a theory by learning one clause at a time and joining those clauses together. A clause is generated either in a top-down, or in a bottom-up fashion. In the former case the starting point is a most general clause that covers all the examples, which is gradually specialized, via a specialization operator, in order to exclude negative examples, while still covering as many positives as possible. In the latter case the starting point is a most specific clause that usually covers a few positive examples, which is gradually generalized using a generalization operator, in order to cover as many positive examples as possible, while excluding the negatives.

Some examples of top-down ILP learners are FOIL [Quinlan, 1990b], HYPER [Bratko, 2001] and TopLoG [Muggleton et al., 2008]. A downside of the top-down approaches is that they are "knowledge-driven", i.e. they rely only on the language bias to guide the search. As a result, they may spend resources in generating a large number of useless clauses that do not cover even a single positive example. In contrast, bottom-up approaches are more "data-driven", i.e. they try to use the training data to identify potentially promising parts of the search space. E.g. Golem [Muggleton et al., 1990], a system based on *relative least-general generalization* (rlgg), generalises pairs of examples creating the least upper bound of the two in the θ-subsumption lattice, each time greedily selecting the clause with the best coverage among a set of rlgg-generated candidates. A downside of bottom-up systems is that they tend to produce overly specific solutions and impose several restrictions to their input language [Corapi, 2012].

A particularly successful family of ILP systems are those that try to combine the best of the two approaches (top-down and bottom-up). Systems like Progol [Muggleton, 1995a] and ALEPH [Srinivasan, 2000] rely on the notion of the *Bottom Clause*. Given a set of mode declarations M and a training example e, a Bottom Clause relative to e, denoted by $\bot_e$, is the most-specific clause of $\mathcal{L}(M)$, up to e.g. a maximum clause length, that covers e. Typically, and provided that target predicate instances are observable in the training data, a Bottom Clause is constructed by selecting such an instance (a positive example e) as a "seed" and placing it in the head of a newly generated clause $\bot_e$ with an empty body. A set of ground atoms that follow deductively from e and the background knowledge are placed in the body of $\bot_e$. Subsequently, constants in $\bot_e$ are replaced by variables, where appropriate, as indicated by mode declarations, and a hypothesis clause is learnt in a top-down fashion, by searching the clauses that θ-subsume $\bot_e$. This strategy allows for a more targeted search, in a space of clauses bounded below by $\bot_e$, where each candidate clause covers at least one positive example (e in particular).

2.4 Learning Programs in the Event Calculus

In this section we present the main challenges related to learning Event Calculus programs with ILP and we discuss the state of the art w.r.t. this task. We present in detail XHAIL, one of the few ILP system that are able to learn theories in the Event Calculus and we discuss XHAIL's scalability issues that we address in the remainder of this thesis.

2.4.1 Non-Observational Predicate Learning

A first difficulty with Event Calculus theories is that the Bottom Clause-based strategy described in Section 2.3.2 cannot be applied directly. The reason is that learning domain-dependent Event Calculus axioms falls in the non-Observational Predicate Learning (non-OPL) class of problems Muggleton [1995b]. In non-OPL, instances of target predicates, that are normally used as seeds for the construction of Bottom Clauses, are not directly observable in the training data. In our case, target predicates are initiatedAt/2 and terminatedAt/2 (see Table 2.3), while the annotation in the training interpretations consists of complex event instances in terms of the holdsAt/2 predicate (see Table 2.2).

A workaround is to use abduction as an intermediate step in Bottom Clause construction, to obtain the missing target predicate instances. Abduction is a mode of logical inference that seeks to extract a set of explanations that make a set of observations true. Abductive Logic Programming (ALP) [Denecker and Kakas, 2002; Kakas et al., 1993; Kakas and Mancarella, 1990] is an abductive framework for logic programming. In ALP the observations are represented by a set of queries, and one derives explanations for these observations in the form of ground facts that make the queries succeed. Formally, an ALP task is a triplet $ALP\langle B, A, E\rangle$, where B is some background theory, A is a set of predicates called *abducibles* and E is a set of observations (ground logical atoms) that must be explained in terms of abducible predicates. The goal is to find an abductive explanation Δ of the observations E, i.e. is a set of abducible predicate instances, such that $B \cup \Delta \vDash E$. In ALP, abductive inference is tightly coupled with Negation as Failure and its semantics [Eshghi and Kowalski, 1989]. Therefore, combining ALP with ILP provides a powerful framework for handling incomplete knowledge in clausal logic. A disadvantage of ALP is that it is an expensive operation, mainly due to the usually very large number of alternative abductive explanations. In an effort to tame the complexity of the task, and also to guide the abductive search towards more meaningful explanations, most ALP implementations use a minimality bias, i.e. they seek for a minimal (and therefore simplest) set of abducible predicates that explain the observations.

Example 2.2. *Assume that we wish to explain the observations (i.e. the annotation) in Table 2.2 in terms of initiatedAt/2 and terminatedAt/2 predicates. Therefore, the set of abducibles for our ALP task is* $A = \{\mathit{initiatedAt}/2, \mathit{terminatedAt}/2\}$. *The background knowledge consists of the axioms of the SDEC (the two axioms in Table 2.1) and all the* *narrative atoms in Table 2.2. Using ALP, we may use the SDEC axioms to explain when the complex events in the observations were initiated or terminated, given that they hold (or not), as stated by the annotation. The smallest set of such explanations is simply*

$$\Delta = \{\mathit{initiatedAt}(moving(id_1, id_2), 1000)\}$$

stating that $moving(id_1, id_2)$ *is initiated at time* 1000 *(since it holds at time* 1001*).*

The abduced atom may now be used to construct a Bottom Clause from the knowledge in Table 2.2. Starting with the clause $\bot = \mathit{initiatedAt}(moving(id_1, id_2), 1000) \leftarrow$ *(a clause with an empty body) and using the mode declarations from Table 2.4, we can construct a bottom clause by adding to the body of* $\bot$ *all atoms that match a body declaration schema and are derivable from the background knowledge, obtaining the clause*

$$\begin{aligned} \bot = \mathit{initiatedAt}(&moving(id_1, id_2), 1000) \leftarrow \\ &\mathit{happensAt}(walking(id_1), 1000), \\ &\mathit{happensAt}(walking(id_2), 1000), \\ &\mathit{holdsAt}(close(id_1, id_2, 25), 1000), \\ &\mathit{holdsAt}(close(id_2, id_1, 25), 1000), \\ &\mathit{holdsAt}(orientation(id_1, id_2, 45), 1000), \\ &\mathit{holdsAt}(orientation(id_2, id_1, 45), 1000). \end{aligned} \tag{2.1}$$

Replacing constants with variables where appropriate, as indicated by mode declarations, we obtain the variabilized Bottom Clause

$$\begin{aligned} \bot' = \mathit{initiatedAt}(&moving(X, Y), T) \leftarrow \\ &\mathit{happensAt}(walking(X), T), \\ &\mathit{happensAt}(walking(Y), T), \\ &\mathit{holdsAt}(close(X, Y, 25), T), \\ &\mathit{holdsAt}(close(Y, X, 25), T), \\ &\mathit{holdsAt}(orientation(X, Y, 45), T), \\ &\mathit{holdsAt}(orientation(Y, X, 45), T). \end{aligned} \tag{2.2}$$

As mentioned in Section 2.3.2, once $\bot'$ *is constructed, a potentially good clause, i.e. one that covers many positive and none of the negative examples may be found by searching the space of clauses that* θ*-subsume* $\bot'$*. Note for instance that the initiatedAt domain-specific axiom of Table 2.3 is derivable from* $\bot'$*, since it* θ*-subsumes* $\bot'$*.*

2.4.2 Problems with Negation as Failure

Another difficulty that arises when learning Event Calculus theories is related to the Negation as Failure operator that Event Calculus uses to model inertia. In this section we discuss the problems that arise in detail.

A first problem is that Negation as Failure is incompatible with the mainstream separate-and-conquer strategy used by the majority of ILP learners, where clauses that cover subsets of the examples are constructed one by one recursively, until all examples are covered. At each step, the positive examples that are covered by a newly generated clause are removed, and the process continues until no more positives are left. However, the non-monotonicity of Negation as Failure makes this strategy essentially unsound. An example, adapted from Ray [2006], follows.

Example 2.3. *Assume that we are given the ILP task $ILP\langle B, E, M\rangle$, where*

$$\begin{aligned}
&B = \{p(X,1) \leftarrow q(X) \land \textit{not}\ r(X), p(X,2) \leftarrow r(X)\}\\
&E^+ = \{p(a,1), p(a,2)\}\\
&E^- = \emptyset\\
&M = \{\mathsf{modeh}(q(+\mathsf{any})), \mathsf{modeh}(r(+\mathsf{any}))\}
\end{aligned} \tag{2.3}$$

i.e. the goal is to induce definitions for the $q/1$ and $r/1$ predicates from the given examples and the background theory. The mainstream, separate-and-conquer ILP approach would start from an empty hypothesis $H = \emptyset$ and the first positive example $p(a,1)$, and then use the first clause from B to abduce the atom $q(a)$. The abduced atom would be generalized to the clause $q(X) \leftarrow$, which would be then added to H. So after the first example is taken into account, $H = \{q(X) \leftarrow\}$. The positive example $p(a,1)$ is now covered by H and thus it is removed. Similarly, in response to the second positive example, $p(a,2)$, the atom $r(a)$ is abduced from the second clause of B, and it is generalized to the new clause $r(X) \leftarrow$, which is also added to H, so $H = \{q(X) \leftarrow,\ r(X) \leftarrow\}$. As previously, the positive example $p(a,2)$ is now covered by H, so it is retracted. Since no more positives remain, learning terminates returning H. However, the second clause in H invalidates the first one, so only one of the positive examples can be covered by H, therefore H is unsound.

A second problem is related to the θ-subsumption-based heuristics used to guide the search in Horn logic, as described in Section 2.3.2, which are known to be inapplicable in general in the case of normal logic programs [Fogel and Zaverucha, 1998]. These heuristics are based on the assumption that "moving up" the subsumption lattice, i.e. from specific to general, increases example coverage, while "moving down", from general to specific, restricts example coverage. However, this does not always hold in normal logic programs, where generalizing (resp. specializing) a clause may result in a hypothesis that covers less (resp. more) examples. An example from the domain of activity recognition follows.

Example 2.4. *Consider the following annotated narrative related to the* fighting *high-level event from CAVIAR:*

Narrative :	**Annotation** :
happensAt$(abrupt(id_1), 1)$.	*not holdsAt*$(fighting(id_1, id_2), 1)$.
happensAt$(abrupt(id_2), 1)$.	*holdsAt*$(fighting(id_1, id_2), 2)$.
holdsAt$(close(id_1, id_2, 23), 1)$.	*holdsAt*$(fighting(id_1, id_2), 3)$.
happensAt$(walking(id_1), 2)$.	
happensAt$(abrupt(id_2), 2)$.	
holdsAt$(close(id_1, id_2, 23), 2)$.	

where $close(X, Y, D)$ *is a statically defined fluent which states that the Euclidean distance between persons* X *and* Y *is less than threshold* D. *Consider also the clauses:*

$$\begin{aligned} C_1 = \textit{initiatedAt}(fighting(X, Y), T) \leftarrow \\ \textit{happensAt}(abrupt(X), T), \\ \textit{not happensAt}(inactive(Y), T), \\ \textit{holdsAt}(close(X, Y, 23), T). \end{aligned}$$

$$\begin{aligned} C_2 = \textit{terminatedAt}(fighting(X, Y), T) \leftarrow \\ \textit{happensAt}(walking(X), T). \end{aligned}$$

$$\begin{aligned} C_2' = \textit{terminatedAt}(fighting(X, Y), T) \leftarrow \\ \textit{happensAt}(walking(X), T), \\ \textit{not holdsAt}(close(X, Y, 23), T). \end{aligned}$$

Clause C_1 *states that* fighting *between two persons* id_1 *and* id_2 *is initiated if one of them exhibits an* abrupt *behavior, the other is not* inactive *and their distance is less than 23 pixel positions on the video frame. Clause* C_2 *states that* fighting *is terminated between two people if one of them walks. Clause* C_2' *is a specialization of* C_2 *and dictates that* fighting *between two persons is terminated when one of them walks away. Consider two hypotheses* H_1, H_2 *where* $H_1 = \{C_1, C_2\}$ *and* $H_2 = \{C_1, C_2'\}$. *Observe that SDEC* $\cup H_1$ *is an incomplete hypothesis, because it does not cover the positive example holdsAt*$(fighting(id_1, id_2), 3)$. *Indeed, by means of clause* C_2 *the fluent fighting*(id_1, id_2) *is terminated at time 2, and thus it does not hold at time 3. On the other hand, hypothesis SDEC* $\cup H_2$ *does cover the positive example at time 3 because clause* C_2' *does not terminate the* fighting *fluent at time 2. We thus have that hypothesis* H_2*, though more specific than* H_1*, covers more examples.*

The shortcomings discussed in this section indicate that mainstream ILP strategies that learn a theory in a clause-by-clause manner cannot by used to induce sound hypotheses in the presence of Negation as Failure. One solution is to employ *theory-level* search [Badea, 2001; Bratko, 1999], i.e. learn a hypothesis in one go, from all available examples at once. In the next section we present the details of XHAIL [Ray, 2009a], an ILP system that learns whole theories and it is also able to address non-OPL problems by combining

Algorithm 1 The XHAIL algorithm
Input: background knowledge B, examples E, mode declarations M, and integer $d > 0$
Output: Hypothesis H

```
***Step 1: Abductive Phase***
**Let** $A_1$ be the set of head mode declarations predicates.
**Let** $\Delta$ be any explanation in $ALP\langle B, A_1, E\rangle$.
***Step 2: Deductive Phase – Kernel Set Generation***
**Let** $A_2$ be the empty set of predicates $\emptyset$.
**Let** $T_2$ be the program obtained by adding to $B$ each fact in $\Delta$.
**for all** facts $\alpha_i \in \Delta$ **do**
    **Let** $m_i$ be any head declaration in $M$ whose schema subsumes $\alpha_i$.
    **Set** $n_i$ to the set of terms in $\alpha_i$ corresponding to $+$ placemarkers in $m_i$.
    **Set** $K_i$ to the fact $\alpha_i$.
    **repeat** up to d times:
        **Let** $Q$ be the set of goals $?type(m)\sigma, schema(m)\sigma$ where $m$ is a body declaration
        and $\sigma$ is a substitution binding all input variables in $m$ to a term in $n_i$
        **Let** $R$ be the set of ground literals of the form $schema(m)\sigma\theta$
        where $schema(m)\sigma\theta$ appears in a goal $G \in Q$ and $\theta$ is an answer substitution
        in $ALP\langle T_2, A_2, G\rangle$.
        **Add** to the body of $K_i$ all literals in $R$ (not already in $k_i$).
        **Add** to $n_i$ all (new) terms in $R$ corresponding to $-$ placemarkers.
    **Let** $K_i'$ be the clause obtained from $K_i$ by replacing all distinct terms corresponding
    to $+$ and $-$ placemarkers with fresh variables.
    **Let** $K_v$ (resp. $K$) be the set of clauses $\{K_1', \ldots, K_n'\}$ (resp. $\{K_1, \ldots, K_n\}$)
***Step 3: Inductive Phase***
**Let** $A_3 = \{use/2\}$
**Let** $T_3$ be the program obtained by adding to $B$ one clause of the form
$\alpha_i \leftarrow use(i,0), try(i,1,v(\delta_i^1)), \ldots, try(i,n,v(\delta_i^n))$ for each clause
$r_i = \alpha_i \leftarrow \delta_i^1, \ldots \delta_i^n \in K_v$
and two clauses $try(i,j,v(\delta_i^j)) \leftarrow use(i,j), \delta_i^j$, $try(i,j,v(\delta_i^j)) \leftarrow \text{not } use(i,j)$
for each literal $\delta_i^j$ in clause $r_i$, where $v(\delta_i^j)$ is a term wrapping the variables of $\delta_i^j$.
**Let** $U$ be any explanation in $ALP\langle T_3, A_3, E\rangle$.
**Let** $H$ be the program obtained from $K_v$ by removing every body atom $\delta_i^j$ for which
the abducible $use(i,j) \notin U$, and removing every clause whose head atom $\alpha_i$ does
not have a corresponding atom $use(i,0)$ in $U$.
**Return** $H$.
```

ILP with ALP, as discussed in Section 2.4.1. We present XHAIL in detail because parts of its methodology are useful throughout this thesis.

2.4.3 The XHAIL system

XHAIL is an abductive-inductive system that constructs hypotheses in a three-phase process. Each of these phases is presented in detail in Algorithm 1, adapted from [Ray, 2009a].

Input

Narrative

happensAt$(abrupt(id_1),\ 1)$.
happensAt$(walking(id_2),\ 1)$.
not holdsAt$(close(id_1, id_2, 23),\ 1)$.
happensAt$(abrupt(id_3),\ 2)$.
happensAt$(abrupt(id_4),\ 2)$.
holdsAt$(close(id_3, id_4, 23),\ 2)$.

Annotation

holdsAt$(fighting(id_1, id_2),\ 1)$.
not holdsAt$(fighting(id_3, id_4),\ 1)$.
not holdsAt$(fighting(id_1, id_2),\ 2)$.
not holdsAt$(fighting(id_3, id_4),\ 2)$.
not holdsAt$(fighting(id_1, id_2),\ 3)$.
holdsAt$(fighting(id_3, id_4),\ 3)$.

Mode declarations (Table 2.4)

Background knowledge: SDEC (Table 2.1)

Phase 1 (Abduction):

$\Delta_1 = \{$initiatedAt$(fighting(id_3, id_4),\ 2)$,
terminatedAt$(fighting(id_1, id_2),\ 1)\}$

Phase 2 (Deduction):

Kernel Set K:

initiatedAt$(fighting(id_3, id_4),\ 2) \leftarrow$
happensAt$(abrupt(id_3),\ 2)$,
happensAt$(abrupt(id_4),\ 2)$,
holdsAt$(close(id_3, id_4, 23),\ 2)$.

terminatedAt$(fighting(id_1, id_2),\ 1) \leftarrow$
happensAt$(abrupt(id_1),\ 1)$,
happensAt$(walking(id_2),\ 1)$,
not holdsAt$(close(id_1, id_2, 23),\ 1)$.

Variabilized Kernel Set K_v:

initiatedAt$(fighting(X, Y),\ T) \leftarrow$
happensAt$(abrupt(X),\ T)$,
happensAt$(abrupt(Y),\ T)$,
holdsAt$(close(X, Y, 23),\ T)$.

terminatedAt$(fighting(X, Y),\ T) \leftarrow$
happensAt$(abrupt(X),\ T)$,
happensAt$(walking(Y),\ T)$,
not holdsAt$(close(X, Y, 23),\ T)$.

Phase 3 (Induction):

Program U_{K_v} (Syntactic transformation of K_v):

initiatedAt$(fighting(X, Y),\ T) \leftarrow$
$use(1, 0), try(1, 1, vars(X, T))$,
$try(1, 2, vars(Y, T))$,
$try(1, 3, vars(X, Y, T))$.

$try(1, 1, vars(X, T)) \leftarrow$
$use(1, 1)$, happensAt$(abrupt(X),\ T)$.
$try(1, 1, vars(X, T)) \leftarrow$ not $use(1, 1)$.

$try(1, 2, vars(Y, T)) \leftarrow$
$use(1, 2)$, happensAt$(abrupt(Y),\ T)$.
$try(1, 2, vars(X, T)) \leftarrow$ not $use(1, 2)$.

$try(1, 3, vars(X, Y, T)) \leftarrow$
$use(1, 3)$, holdsAt$(close(X, Y, 23),\ T)$.
$try(1, 3, vars(X, T)) \leftarrow$ not $use(1, 3)$.

terminatedAt$(fighting(X, Y),\ T) \leftarrow$
$use(2, 0), try(2, 1, vars(X, T))$,
$try(2, 2, vars(Y, T))$,
$try(2, 3, vars(X, Y, T))$.

$try(2, 1, vars(X, T)) \leftarrow$
$use(2, 1)$, happensAt$(abrupt(X),\ T)$.
$try(2, 1, vars(X, T)) \leftarrow$ not $use(2, 1)$.

$try(2, 2, vars(Y, T)) \leftarrow$
$use(2, 2)$, happensAt$(walking(Y),\ T)$.
$try(2, 2, vars(Y, T)) \leftarrow$ not $use(2, 2)$.

$try(2, 3, vars(X, Y, T)) \leftarrow$
$use(2, 3)$, not holdsAt$(close(X, Y, 23),\ T)$.
$try(2, 3, vars(X, Y, T)) \leftarrow$ not $use(2, 3)$.

Search:

$ALP($SDEC $\cup\ U_{K_v}, \{use/2\}, Narrative \cup Annotation)$

Abductive Solution:

$\Delta_2 = \{use(1, 0), use(1, 3),$
$use(2, 0), use(2, 2)\}$

Output hypothesis

initiatedAt$(fighting(X, Y),\ T) \leftarrow$
holdsAt$(close(X, Y, 23),\ T)$.

terminatedAt$(fighting(X, Y),\ T) \leftarrow$
happensAt$(walking(Y),\ T)$.

TABLE 2.5: Hypothesis generation by XHAIL for Example 2.5.

Given an ILP task $ILP\langle B, E, M\rangle$, the first two phases of the XHAIL algorithm return a ground program K, called *Kernel Set of* E, such that $covers(B \cup K, E)$. The first phase generates the heads of K's clauses by abductively deriving from B a set Δ of instances of head atoms, as defined by the language bias M, such that $covers(B \cup \Delta, E)$. The second phase generates the Kernel Set K, by forming clauses with a previously abduced atom in the head, and instances of body declaration atoms that deductively follow from $B \cup \Delta$ in the body.

The Kernel Set is a "multi-clause equivalent" [Ray, 2005, 2006; Ray et al., 2004] of the Bottom Clause. That is, just as a Bottom Clause is a most-specific clause that covers a single positive example, the Kernel Set is, by construction, a most-specific program that covers all the positive examples. Its purpose in the induction process is similar to that of a Bottom Clause: To provide a syntactic and semantic bias that delimits a part of the search space that is likely to contain hypotheses. Just as ILP algorithms that learn from Bottom Clauses use a clause refinement operator to generate and test clauses that θ-subsume the Bottom Clause, XHAIL uses a *theory refinement operator*, that generates and tests candidates in the space of theories that θ-subsume the Kernel Set, in an effort to find a hypothesis that covers all the positive and none of the negative examples.

To this end, the Kernel Set is first variabilized, i.e. constants are replaced by variables where appropriate, as indicated by the mode declarations. The variablized Kernel Set, K_v, is subject to a syntactic transformation of its clauses, which involves two new predicates $try/3$ and $use/2$. For each clause $r_i \in K_v$, where $1 \leq i \leq |K|$, and each body literal $\delta_i^j \in r_i$, where $1 \leq j \leq |body(r_i)|$, a new atom $v(\delta_i^j)$ is generated, as a special term that contains the variables that appear in δ_i^j. The new atom is wrapped inside an atom of the form $try(i, j, v(\delta_i^j))$. An extra atom $use(i, 0)$ is added to the body of C_i and two new clauses $try(i, j, v(\delta_i^j)) \leftarrow use(i, j), \delta_i^j$ and $try(i, j, v(\delta_i^j)) \leftarrow \text{not } use(i, j)$ are generated, for each body literal $\delta_i^j \in C_i$.

All these clauses are put together into a program U_{K_v}. U_{K_v} serves as a "defeasible" version of K_v from which literals and clauses may be selected, or discarded, in order to construct a hypothesis that accounts for the examples. This is realized by solving an ALP task with $use/2$ as the only abducible predicate. The intuition behind this transformation is as follows: In order for the head atom of clause $C_i \in U_{K_v}$ to contribute towards the coverage of an example, each of its $try(i, j, v(\delta_i^j))$ atoms must succeed. By means of the two rules added for each such atom, this can be achieved in two ways: Either by assuming $\text{not } use(i, j)$, or by satisfying δ_i^j and abducing $use(i, j)$. A hypothesis clause is constructed by the head atom of the i-th clause r_i of K_v, if $use(i, 0)$ is abduced, and the j-th body literal of r_i, for each abduced $use(i, j)$ atom. All other clauses and literals from K_v are discarded. Search is biased by minimality, i.e. preference towards hypotheses with fewer literals. This is realized by means of abducing a minimal set of $use/2$ atoms.

Example 2.5. *Table 2.5 presents the process of hypothesis generation by XHAIL. The input consists of a set of examples, a set of mode declarations (presented in Table 2.4) and the axioms of the SDEC as background knowledge. The annotation says that $\mathit{fighting}$ between persons id_1 and id_2 holds at time 1 and it does not hold at times 2 and 3, hence it is terminated at time 1. Respectively,* fighting *between persons id_3 and id_4 holds at time 3 and does not hold at times 1 and 2, hence it is initiated at time 2. XHAIL obtains these explanations for the holdsAt/2 literals of the annotation abductively, using the head mode declarations as abducibles. In its first phase, it derives the two ground atoms in Δ_1 (Phase 1, Table 2.5). In its second phase, XHAIL forms a Kernel Set (Phase 2, Table 2.5), by generating one clause from each abduced atom in Δ_1, using this atom as head, and body literals that deductively follow from* SDEC $\cup$ Δ_1 *as the body of the clause.*

The Kernel Set is variabilized and the third phase of XHAIL, i.e. the actual search for a hypothesis starts. This search is biased by minimality, *i.e. preference towards hypotheses with fewer literals. A hypothesis is thus constructed by dropping as many literals and clauses from K_v as possible, while correctly accounting for all the examples. The syntactic transformation on K_v results in the defeasible program U_{K_v}.*

Literals and clauses necessary to cover the examples are selected from U_{K_v} by means of abducing a set of $use/2$ atoms, as explanations of the examples, from the ALP task presented in Phase 3 of Table 2.5. Δ_2 from Table 2.5 is a minimal explanation for this ALP task. $use(1, 0)$ and $use(2, 0)$ correspond to the head atoms of the two K_v clauses, while $use(1, 3)$ and $use(2, 2)$ correspond respectively to their third and second body literal. The output hypothesis in Table 2.5 is constructed by these literals, while all other literals and clauses from K_v are discarded.

By means of abduction, XHAIL is able to learn definitions for unobserved predicates, thus dealing with non-Observational Predicate Learning. Moreover, thanks to the non-monotonic semantics of abduction in ALP, and the fact that XHAIL bases its entire functionality on abduction (note that XHAIL "implements" deductive and inductive inference by means of an abductive process), XHAIL can learn Negation as Failure and ensure soundness in non-monotonic learning tasks. Therefore, it can fully address the problem of learning event definitions as logic programs in the Event Calculus. Its major drawback, however, is that it scales poorly. The reason is that XHAIL employs a theory-level refinement operator that learns whole hypotheses from all the training data at once. This is necessary in order to overcome the difficulties, discussed earlier in this section, that standard sequential covering ILP algorithms face when learning in the presence of Negation as Failure. The downside is the increased combinatorial complexity of jointly optimizing all clauses in a hypothesis.

2.5 Related Work

In this section we discuss relevant work from the literature on a number of related ILP approaches. This discussion is divided into two parts: In the first part (Section 2.5.1), we present an overview of ILP systems that, like XHAIL, attempt to learn normal logic programs with unobserved target predicates. In the second part (Section 2.5.2), we present a more general discussion on the task of learning temporal theories with ILP. Some applications to event recognition tasks are included in this discussion, while the rest of the presented approaches are judged to be relevant, due to temporal aspects of the learning domains and the learnt theories.

2.5.1 Related ILP Systems

Several ILP approaches have addressed the problem of learning normal logic programs [Ade and Denecker, 1995; Bain and Muggleton, 1990; Bergadano et al., 1996; Dimopoulos and Kakas, 1995; Martin and Vrain, 1996; Nicolas and Duval, 2001; Sakama, 1999; Seitzer, 1997]. However, as pointed out in [Ray, 2009a], none of these approaches is general enough to learn Event Calculus theories, since they are capable of Observational Predicate Learning only. Also, they assume restrictions on the use of Negation as Failure, for instance, they are able to learn normal logic programs, but not with a normal logic program as background knowledge.

Similar restrictions hold for more recent approaches. For instance, PROGOL-5 [Muggleton and Bryant, 2000b] is an extension of PROGOL that uses the technique of theory completion to perform a form of abduction. However the so-called "contra-positive" method on which theory completion relies does not work in the presence of Negation as Failure [Ray, 2005, 2006; Ray et al., 2003]. A similar limitation holds for a number of Inverse Entailment-based systems, like ALECTO, HAIL, CF-Induction [Inoue, 2004], Residue Hypothesis Finding [Yamamoto, 2003] and the SOLAR system [Nabeshima et al., 2003] for consequence finding: They are all capable of handling non-Observational Predicate Learning via some form of abductive reasoning, but not with a normal program as background knowledge.

IMPARO [Kimber et al., 2009] uses a proof procedure called Induction on Failure, that handles the non-Observational Predicate Learning problem, but it is also limited to Horn logic. Induction on Failure has been generalized to the case of full clausal logic in [Kimber, 2012]. However, the work in [Kimber, 2012] (as well as that of [Kimber et al., 2009]) focuses on theoretical results that address limitations of PROGOL's Inverse Entailment proof procedure, as identified first in [Yamamoto, 1997], related to its incompleteness (hypotheses that cannot be found by Inverse Entailment). No experimental evaluation of Induction on Failure is presented in [Kimber, 2012], while in the same work it is reported, based on empirical observations, that significant work is required in order for this proof procedure to be made efficient for any practical purposes [Kimber, 2012].

In contrast to the above-mentioned approaches, a family of abductive-inductive systems has been proposed that is capable of solving the same class of problems as the XHAIL system (and therefore these systems are capable of learning theories in the Event Calculus). TAL [Corapi et al., 2010] is the first system in this family. It is a top-down non-monotonic learner that works by appropriately mapping an ILP problem to a corresponding ALP instance, so that solutions for the latter may be translated to solutions for the initial ILP problem. Given an ILP task $\mathcal{I}$, the machinery behind TAL involves three steps: The first step establishes a transformation that allows to represent every possible clause r in a potential inductive hypothesis for $\mathcal{I}$ as a single fact, that references the mode declarations that are used to form the clause r. A second step uses the mode declarations themselves to define a so-called "top theory", a concept introduced in [Muggleton et al., 2008]. Intuitively (and omitting technical details), a top theory is a mode-declarations-based meta-theory that allows to generate hypotheses from the space defined by the mode declarations. In a third step, an abductive task $\mathcal{A}$ is instantiated on this top theory. The goals in this task are the training examples and the abducibles are the predicates used in step 1, to codify candidate clauses for $\mathcal{I}$ as facts that reference their generating mode declarations. Then learning an inductive hypothesis for the original ILP task $\mathcal{I}$ is translated into finding an abductive solution for $\mathcal{A}$: An abduced atom may be used as a set of prescriptions for forming a clause by properly combining the mode declaration specifications that are codified in this atom.

TAL uses Prolog's SLDNF resolution to realize the abductive process. Its ideas were employed in the ASPAL system [Corapi et al., 2012], an ILP learner which relies on Answer Set Programming as a unifying abductive-inductive framework. Interesting as it may be, the approach of TAL and ASPAL ultimately faces the same problems as that of XHAIL, at least with respect to scalability: To ensure soundness in the presence of Negation as Failure, TAL must learn a hypothesis in one shot, from the entirety of training examples, resulting to an intractable search space.

This issue is addressed to some extent, in [Athakravi et al., 2013], where the methodology of TAL and ASPAL has been ported into a learner that constructs hypotheses progressively, towards more scalable learning. To address the fact that the top theory of TAL/ASPAL grows exponentially with the length of its clauses, causing a grounding bottleneck, RASPAL, the system proposed in [Athakravi et al., 2013], imposes bounds on the length of the top theory. Partial hypotheses of specified clause length are iteratively obtained in a refinement loop. At each iteration of this loop, the hypothesis obtained from the previous refinement step is further refined by dropping or adding literals or clauses, using theory revision as described in [Corapi et al., 2008]. The process continues until a complete and consistent hypothesis is obtained. However, in order to ensure soundness, RASPAL still has to process all examples simultaneously. At each iteration of its refinement loop, all examples are taken into account repeatedly, in order to ensure that the revisions account for all them. Therefore, the grounding bottleneck that RASPAL faces is expected to persist in domains that involve large volumes of sequential data, typical of temporal applications, as the ones that we address in this work. This is because even by imposing a small initial maximum clause length to RASPAL, in order to constrain the search space, with a sufficient amount of data the resulting ground program will still be intractable, if the data is processed simultaneously.

In [Law et al., 2014], a framework for learning in Answer Set Programming (ASP) is introduced. This framework, call ASILP, extends previous related work on learning from partial interpretations [De Raedt, 1997; Inoue et al., 2014], i.e. incomplete examples in the Learning from Interpretations setting, as well as learning from stable models [Otero, 2001; Sakama and Inoue, 2009] in the presence of incomplete information. The aim of this extension is to obtain a general-purpose methodology for learning ASP programs with ILP. ASILP learns a hypothesis H in the presence of background knowledge B, that allows observations to be "bravely entailed", i.e. to be true in some, but not all, the stable models of $B \cup H$. The rationale follows an argument from [Sakama and Inoue, 2009], where it is argued that such a framework (called brave induction), should be preferred over the regular ILP setting, where an acceptable hypothesis H should be such that the observations are true in all stable models of $B \cup H$. The reason in that the regular ILP framework is often "too strong", particularly in cases with indefinite/incomplete information. Thanks to its connection to ASP and the stable model semantics, ASILP can fully handle non-Observational Predicate Learning and Negation as Failure. However, as pointed out in [Law et al., 2014] ASILP does not scale adequately.

The combination of ILP with ALP has recently been applied to *meta-interpretive learning* (MIL), a learning framework where the goal is to obtain hypotheses in the presence of a meta-interpreter. The latter is a higher-order program, referencing predicates or even rules of the domain. Given such background knowledge and a set of examples, MIL uses abduction w.r.t. the meta-interpreter to construct first-order hypotheses. MIL can be realized both in Prolog and in Answer Set Programming, and it has been implemented in the METAGOL system [Muggleton et al., 2014]. MIL is an elegant framework, able to address difficult problems like predicate invention and mutually recursive programs. However, it has a number of important drawbacks. First, its expressivity is limited, as MIL is currently restricted to *dyadic Datalog*, i.e. Datalog where the arity of each predicate is at most two. Second, given the increased computational complexity of higher-order reasoning, scaling to large volumes of data becomes a potential bottleneck.

2.5.2 Logical Learning in Temporal Domains

The Chronicle Recognition System (CRS) [Dousson and Le Maigat, 2007; Ghallab, 1996] is a temporal reasoning system that allows for efficient and scalable temporal reasoning. CRS has been used for event recognition in various domains, including medical applications and computer network management [Callens et al., 2008b; Carrault et al., 2003; Dousson and Le Maigat, 2007]. A chronicle is a temporal constraint network [Dechter et al., 1991] (although a Petri nets-based semantics has also been formulated for CRS [Choppy et al., 2009]). It can be seen as a definition of a high-level event that links together a set of low-level events via a set of temporal (and possibly atemporal) constraints. Chronicle high-level event specifications can be expressed in an ad-hoc first order language. Moreover high-level event patterns in the CRS language may be learnt with ILP. For instance, in [Carrault et al., 2003] the authors use the ICL system [De Raedt and Van Laer, 1995] to learn a set of first order temporal clauses for characterization of cardiac arrhythmia, based on training interpretations extracted from a time series. These clauses are subsequently translated into CRS language rules.

An interesting ILP application in a challenging temporal domain is presented in [Badea, 2000], where PROGOL is used to learn a set of first-order stock market trading rules in the form of "buy/sell" policies from historical market time series. Each trading rule defines a pattern of temporal combinations of stock trading technical indicators (e.g. moving averages, the Relative Strength Index, the Average Directional Movement Index, stochastic oscillators and so on) and is able to indicate promising buying/selling opportunities by matching this pattern on temporal incoming data.

In [Dubba et al., 2010], an ILP approach is presented that uses videos recorded in an airport to learn definitions of interesting high-level events in the form of first order temporal clauses. Spatio-temporal relations between objects are extracted from video footage, forming a set of time-stampled training interpretations, which are subsequently properly annotated with events of interest. Then, standard learning-from-interpretations ILP techniques are used to extract a set of high-level event definitions. The learnt event patterns include definitions for *rear loading/unloading*, *aircraft arrival/departure*, *jet bridge positioning* and so on. In [Dubba et al., 2011, 2015], this work is extended with the addition of a spatial logical calculus [Randell et al., 1992] that allows abstracting and reasoning about quantitative spatial information, and also by interleaving induction with abduction to improve the learning outcome. This interleaved framework learns in a set cover loop. At each iteration, an induced clause r is added to the background knowledge B and all positive examples covered by $B \cup r$ are removed. Then $B \cup r$ is used to abductively "guess" a set of explanations for the remaining positive examples. A scoring scheme based on a notion of "distance" between positive examples and such abductive explanations is defined and the examples that are "close enough" to the explanations are considered covered and are removed from the yet-to-be-covered positives. The loop continues until all positives are covered.

In [Needham et al., 2005], the PROGOL ILP system is used to learn protocols of simple table top games – a simple card game called "snap", as well as "paper-scissors-rock" – from real sensory data originating form a video camera and a microphone, collected while two persons were playing these games. Game protocols learnt by PROGOL are in the form of actions that should be executed at particular times, given a state of the game at that time. A more recent similar approach is included in [Laguna, 2014], where the TILDE ILP system is used to learn first-order theories for activity recognition, using training interpretations compiled from videos of an internal space.

Common to the aforementioned approaches is the fact that the temporal theories they induce cannot reason about the effects of event occurrences. In contrast, formalisms described collectively with the term "temporal action theories" [Gelfond and Lifschitz, 1993, 1998] are designed for that particular type of reasoning. As a result, a substantial amount of work exists for learning such theories from narrative histories describing a system's dynamic behavior. Early attempts include [Moyle and Muggleton, 1997] and [Moyle, 2002], where ILP techniques are used to learn domain-specific axioms in the Event Calculus. The work in [Moyle and Muggleton, 1997] relies on *theory completion* [Mueller, 2008; Muggleton and Bryant, 2000b] for eliminating initiatedAt and terminatedAt predicates, and re-writing the Event Calculus axioms in terms of a single "flips" predicate, thus effectively nullifying the non-monotonic effects of the Negation as Failure operator. Then PROGOL is used to learn a definition for the "flips" predicate. In [Moyle, 2002], the ALECTO ILP system is used to learn a set of domain-specific axioms in the Event Calculus representing a set of robot navigation instructions. ALECTO is an extension of PROGOL that uses SOLD resolution [Yamamoto, 2000] as an abductive procedure, via which it abduces instances of initiatedAt and terminatedAt predicates and then uses PROGOL to learn domain-specific axioms from these instances and the narrative.

The approaches in [Moyle and Muggleton, 1997] and [Moyle, 2002] are restrictive in several ways. In addition to the re-writing of the Event Calculus axioms in terms of the "flips" predicate, the work in [Moyle and Muggleton, 1997] also requires extra artificial constraints that specify time intervals in which fluents are not clipped [Moyle, 2003; Ray, 2009a]; Also, in order to learn in the presence of Negation as Failure in the background knowledge, the approach in [Moyle, 2002] relies on PROGOL's monotonic induction technique, which, as was pointed out in [Lorenzo and Otero, 2000; Otero, 2001; Sakama, 2000, 2005], does not extend well to normal programs.

A number of approaches attempted to learn temporal action theories using entirely monotonic methods [Könik and Laird, 2006; Lorenzo, 2002; Lorenzo and Otero, 2000; Otero, 2003, 2004; Rodrigues et al., 2010a,b, 2011]. These approaches modify the underlying formalisms by replacing inertial axioms, which involve non-monotonic operators, by frame axioms, i.e. rules that explicitly specify the "non-effects" of events/actions, and use standard monotonic ILP techniques for learning from "complete narratives" [Inoue et al., 2005] (i.e. narratives that contain explicit truth values for fluents at each time point). However, this approach is hardly a solution, since it introduces the well-known frame problem [Gelfond and Lifschitz, 1993; McCarthy, 1986] into the learning task, which action formalisms were created to resolve at the first place, [Inoue et al., 2005; Ray, 2009a]. All these difficulties are successfully handled by more recent non-monotonic ILP learners, such as XHAIL and TAL [Corapi et al., 2010; Ray, 2009a].

Relational learning techniques have also been used in the temporal domain of process mining [Van der Aalst et al., 2003]. In process mining applications the input consists of logs, i.e. sets of sequences of time stamped actions (traces), collected within a period of time. The goal is to learn a set of interesting processes, i.e. temporally ordered sets of actions that capture some particular business logic. Although the dominant paradigm in process mining has been the discovery of procedural process models, like Petri nets [Van Dongen et al., 2009] or Event-driven Process Chains [Van Dongen and Van der Aalst, 2004], logic-based methods have been attracting attention, thanks to their ability to induce declarative processes in the from of informative first-order rules that describe dependencies between actions. In [Chesani et al., 2009] the authors use a logic programming-based framework called $\mathcal{S}$CIFF [Alberti et al., 2008], to represent actions, and an ILP system to learn first-order process patterns from action traces in the form of $\mathcal{S}$CIFF rules. The ontology of the $\mathcal{S}$CIFF formalism comprises events (occurrences of actions in time) and positive (resp. negative) *expectations*, i.e. a special type of event that is expected to happen (resp. not happen) in the near future, once a sequence of normal events has taken place. $\mathcal{S}$CIFF rules comprise events in the body and conjunctions or disjunctions of expectations in the head. A set of $\mathcal{S}$CIFF rules defines a set of interdependent constraints on the actions (and their order) in the domain. In [Chesani et al., 2009], a training interpretation is formed from each action trace and each interpretation is annotated as positive or negative, based on whether or not it complies to a particular business logic. This input is passed to the ICL system [De Raedt and Van Laer, 1995], which learns a $\mathcal{S}$CIFF theory representing a process.

In [Cattafi et al., 2010], the $\mathcal{S}$CIFF framework is combined with the Declarative Process Model Learner (DPML) [Lamma et al., 2007], a simplification of the ICL system tailored for inducing logical process models, to address the problem of incremental learning of process models, where new logs arrive over time and the current process model must account for them. The resulting system uses a set of standard ILP revision operators to modify current process models in the face of new evidence. The same problem (revising process models) is also addressed in [Maggi et al., 2011], this time within the framework of non-monotonic ILP [Sakama, 2001]. In particular, [Cattafi et al., 2010] uses TAL

[Corapi et al., 2010] to revise a process model within the framework of “theory revision as non-monotonic ILP” [Corapi, 2012; Corapi et al., 2011a, 2008; Maggi et al., 2011] (this framework is discussed in detail in Section 3.1 of this thesis).

Requirements engineering [Dardenne et al., 1993] refers to the task of identifying and analyzing particular specifications and stakeholder requirements during software development. In such a context, extracting formal requirements from a set of high-level, intuitive descriptions is a challenging task. In [Alrajeh et al., 2006, 2009], the XHAIL system is used for extracting requirements from positive and negative examples, representing respectively descriptions of desirable and undesirable system behaviour over time. The input also comprises some initial (but incomplete) requirement specifications in the form of domain-specific background knowledge. The goal is to learn a logical theory representing additional requirement specifications, so that the complete requirements' set covers all positive but none of the negative scenarios. The approach in [Alrajeh et al., 2006, 2009] uses a translation of Linear Temporal Logic [Pnueli and Manna, 1992], a formal specification language used by software engineers, to the Event Calculus, and then uses XHAIL to learn formal specifications in the form of an Event Calculus theory. Closely related work is that in [Alrajeh et al., 2011], where XHAIL is used to learn specifications, in the Event Calculus, for Modal Transition Systems, a formalism for modeling and reasoning about the behavior of a system over time; [Corapi et al., 2011a] and [Corapi et al., 2011b], where TAL [Corapi et al., 2010] is used to learn normative frameworks for virtual societies in the form of Event Calculus programs; and [Corapi et al., 2008], where XHAIL is used to learn mobile phone user preferences from historical traces of her behavior over short periods of time, it the form of Event Calculus programs.

Another important field where temporal relational learning has been used is systems biology. In [Tamaddoni-Nezhad et al., 2006, 2007], a combination of abduction and induction is used, to model inhibition between enzymes in metabolic networks (networks of interlinked chemical reactions that take place during the metabolism process). The goal is to derive formal models for inhibition prediction in such networks, which may be used to identify drugs that may have undesirable inhibitory effects on other enzymes in a metabolic network. To this end, PROGOL-5 [Muggleton and Bryant, 2000a] is used, which is capable for abductive reasoning to some extent (in particular, in domains with no negation in the background knowledge), to adbuce a set of ground facts about inhibited enzymes from observations of metabolite concentration. The abduced atoms are subsequently generalized to obtain a set of lifted first-order rules describing inhibition in the metabolic network. The work in [Tamaddoni-Nezhad et al., 2006, 2007] builds upon earlier results on modeling inhibition in metabolic networks [Tamaddoni-Nezhad et al., 2004] by taking into account important temporal aspects of the observations. However, as the authors in [Tamaddoni-Nezhad et al., 2006] point out, the temporal model used in that work is over-simplified and transition to a more robust model based on the Event Calculus-based is desirable in the particular biological domain.

The same problem of learning logical inhibition models has been addressed in several other works that improve on the approach of [Tamaddoni-Nezhad et al., 2006, 2007]. In [Demolombe et al., 2013] the authors use SOLAR for learning in this domain, an ILP system with superior abductive capabilities compared to PROGOL-5, which was used in [Tamaddoni-Nezhad et al., 2006, 2007]. XHAIL has also been used for learning relations in metabolic networks in [Bragaglia and Ray, 2015; Ray, 2009b; Ray et al., 2010], offering better handling of negation and abduction, while in [Ray and Bryant, 2008] it is used to infer gene functions from mutations. The SOLAR system has also been used for completing metabolic networks by discovering missing causal relations [Inoue et al., 2013] with meta-level abduction. The CF-Induction abductive-inductive framework of the IMPARO system has also been applied for learning relational models from metabolic networks [Doncescu et al., 2007; Yamamoto et al., 2008, 2010].

In the field of systems biology, a domain of particular interest is large-scale modeling of biochemical processes, in order to predict the behavior of a complex biological system under various circumstances [Calzone et al., 2006]. ILP methods have been used in this domain, which is inherently temporal. [Calzone et al., 2006; Fages and Soliman, 2008] introduced BIOCHAM, a Datalog-based, domain-specific language for reasoning about properties and dynamics of biochemical processes, and used an ILP approach to learn relational models of such processes. Each BIOCHAM entity is associated with a boolean variable representing its presence or absence in the system. Reaction rules are then interpreted as an asynchronous transition system over states defined by the vector of boolean variables. The Computation Tree Logic [Clarke et al., 1999] (CTL) was used to specify temporal interactions between BIOCHAM entities. A set of first-order rules capturing the dynamics of a biochemical process was learnt by providing a CTL specification of temporal constraints in the system and searching for good rules, i.e. rules whose most of their groundings satisfy these constraints. Closely related is [Synnaeve et al., 2011], where the SOLAR aductive-inductive learner is used to learn first-order models of biochemical processes from temporal data.

2.6 Summary

In this chapter we presented the basics of the Event Calculus formalism that we use in this thesis, in addition to some necessary background on ILP. We analyzed, by means of concrete examples, the challenges involved in learning Event Calculus theories with ILP, namely, handling Negation as Failure in the background knowledge and learning definitions for predicates (initiatedAt and terminatedAt) that are not directly observable in the input to the learning task. We then presented and discussed in detail the XHAIL system, an ILP system that overcomes these difficulties via a combination of ILP techniques with Abductive Logic Programming.

We also discussed related work on similar ILP approaches that combine abductive and inductive reasoning and are also able, to some extent, to address the same issues. We pointed out the limitations of all such approaches, that ultimately render them insufficient for the learning task that we address in this thesis: Most of these approaches are not general enough for the particular task, since they are capable of handling unobserved predicates via abductive reasoning, but not in the general case of normal logic programs, where Negation as Failure is allowed both in the background knowledge and in the bodies of hypothesized clauses. A few approaches that are able to successfully overcome both obstacles, like XHAIL itself, the systems of the TAL family (TAL, ASPAL and RASPAL), ASILP and METAGOL do not scale to the data volumes of event recognition applications.

Additionally, we discussed related work on learning first-order temporal theories with ILP, as well as applications to particular domains that require temporal reasoning, like process mining, requirements engineering, systems biology and event recognition. These approaches fall into two main categories: Those that utilize some temporal formalism, like the Event Calculus, or the Situation Calculus, to reason about the effects of events on properties of the modeled domain, and those that do not. Approaches in the former category suffer from the limitations outlined above: They cannot fully address all aspects involved in learning such temporal formalisms and as a result, they often resort to ad-hoc workarounds that are mostly sub-optimal, while they do not scale to large data volumes. On the other hand, approaches in the latter category lack a domain-independent methodology for reasoning about the evolution of a system in time, and therefore, they are neither general, nor robust enough to be used in applications, like event recognition, where this type of reasoning is necessary in principle.

3 ILED: Incremental Learning of Event Definitions

In the previous chapter we described the XHAIL system and showed that it provides an appropriate framework for learning event definitions in the form of domain-specific axioms in the Event Calculus. As mentioned in Section 2.4.3, however, its main limitation is that it scales poorly, due to its theory-level refinement operator that learns whole hypotheses from all the training data at once, which results in a search space of increased combinatorial complexity. Furthermore, XHAIL is designed for inducing sound hypotheses (i.e. hypotheses that cover all the positive and none of the negative examples). This makes inappropriate any type of heuristic search (e.g. a search strategy aiming to cover/reject a good portion of the positive/negative examples respectively), leaving no other option but a complete search in the space of theories that θ-subsume the Kernel Set. The hardness of this task increases with the size of the training data, since the increase in the number of domain constants results in a combinatorial explosion in the size of the program obtained by fully grounding the search space that the Kernel Set defines, a necessary step in the Learning from Interpretations setting.

In this chapter, we address the problem of scaling-up the basic XHAIL methodology. We do so by proposing an *incremental learning setting* [Langley, 1995; Maloof and Michalski, 2004; Utgoff, 2011], where the XHAIL functionality is adapted to work with training examples that arrive over time. The resulting system, called ILED (Incremental Learning of Event Definitions) [Katzouris et al., 2015], is designed to induce sound theories (as XHAIL itself) and scales well beyond the data volumes that XHAIL can handle. Central to our approach is the process of *theory revision* [Wrobel, 1994, 1996], where an initial hypothesis, obtained from the first incoming batch of examples, is gradually revised to account for new training instances. In contrast to sequential covering algorithms, which are designed for Horn logic and discard each positive example once it is covered by a gradually constructed hypothesis, ILED preserves past examples in an external database, called *historical memory*. By periodically accessing these examples and evaluating candidate revisions on them, ILED is able to avoid cases like the one illustrated in Example 2.3 where modifications to a hypothesis may invalidate parts of the hypothesis that remain unmodified.

ILED subsumes the core XHAIL machinery, i.e. it learns/revises whole theories using an abductive search procedure to generalize Kernel Sets. However, this search procedure remains overall tractable, by learning from relatively small data chunks. Typically, in each of these chunks, the number of domain constants is small, ensuring that the cost of ILED's expensive revision process remains low w.r.t. each chunk. Moreover, by means of the *support set*, a compressive memory structure that encodes the coverage, in the historical memory, of each clause in the running hypothesis, ILED is ensured to learn a sound hypothesis with no more than $2n$ revisions, where n is the number of data chunks.

To summarize, while XHAIL learns a theory in one go from all the training data at once, ILED gradually revises an initial theory on data chunks that arrive over time. And while XHAIL deals with an intractable search space, even with relatively small data volumes, ILED needs a linear, in the size of the training set, number of relatively "cheap" operations to learn a hypothesis, while ensuring soundness of the outcome. It therefore significantly scales up the XHAIL algorithm. Moreover, we present experiments which demonstrate that ILED learns hypotheses of comparable quality (w.r.t. predictive accuracy and size) to those of XHAIL's.

In the remainder of this chapter the details of the ILED system are presented, starting with some necessary background on theory revision and incremental learning. We then provide a formal definition of the underlying incremental learning setting, define the support set, give an overview of ILED's strategy and prove its soundness and the upper bound in the number of revisions required to compute a hypothesis. Finally, we present the details of ILED's theory revision technique and conclude with a discussion on some of ILED's features and limitations.

3.1 Theory Revision and Incremental Learning

In this section we present some necessary background on theory revision and incremental learning. Theory revision [Wrobel, 1996] is the task of modifying imperfect hypotheses to improve their quality. Initially, theory revision systems were developed with the goal to improve a roughly correct hypothesis in the face of new examples that arrive over time [Adé et al., 1993; Ade et al., 1994; Richards and Mooney, 1995, 1991; Wogulis and Pazzani, 1993]. Over the years, theory revision systems became able to induce new hypotheses from scratch, which may subsequently be refined w.r.t. new training data [Corapi et al., 2008; D. Raedt, 1992; Esposito et al., 1996, 2000]. As a result, in addition to a means of improving an input hypothesis, in more recent work [Muggleton et al., 2012; Paes et al., 2007], theory revision systems are also considered as learning systems that exploit previous computations to speed-up the learning. This is important, since revising a hypothesis is generally considered more efficient than learning it from scratch [Biba et al., 2006a; Cattafi et al., 2010; Esposito et al., 2000].

A hypothesis H is called *incomplete* if it does not account for some positive examples and *inconsistent* if it erroneously accounts for some negative examples. Theory revision systems use a set of revision operators to act upon a hypothesis and alter the set of examples it accounts for. Incompleteness is typically handled by a generalization operator, that e.g. adds new clauses, or removes literals from existing clauses. Inconsistency is handled by a specialization operator, that e.g. removes clauses, or adds new literals to existing clauses.

In an incremental learning setting examples arrive over time, contrary to a *batch* setting, where all examples are available from the start [Biba et al., 2006b; Di Mauro et al., 2005, 2004; Esposito et al., 2004; Giraud-Carrier, 2000; Langley, 1995; Mooney, 1992; Semeraro et al., 1997; Westendorp, 2002]. Occasionally, in ILP literature the term "incremental" is used in a different way (e.g. see Flach [1998]; Ray [2005, 2006]; Ray and Inoue [2007]), to describe a property (e.g. "incremental theory construction") of an otherwise batch learning setting. Sequential covering algorithms belong in this category, having in place all training examples when learning commences, but processing them one by one. In what follows, by "incremental learning" we denote a setting where examples arrive over time and learning relies on theory revision.

Of particular interest for the purposes of our work is the so-called *full memory* incremental setting [Biba et al., 2006b]. In this setting, examples are stored in an external database (historical memory) upon their arrival and revisions to the hypothesis at hand in the face of new evidence must account for the entire historical memory as well. The main challenge of a full memory approach is to scale-up to a growing size of experience. This is in line with a key requirement of incremental learning where "the incorporation of experience into memory during learning should be computationally efficient, that is, theory revision must be efficient in fitting new incoming observations" [Di Mauro et al., 2005; Langley, 1995].

3.2 The ILED System

In this section we present the ILED system. ILED starts with an empty hypothesis $H_n = \emptyset$ and an empty database of examples $\mathcal{E} = \emptyset$. It learns a hypothesis H_1 from the first available example w_1 that arrives, which is subsequently stored in $\mathcal{E}$. As new examples arrive, it gradually revises the hypothesis at hand so that it accounts both for the new examples and the accumulated experience in the historical memory. Definition 3.1 formalizes our incremental learning setting. Recall from Section 2.2.1 that SDEC in what follows denotes the Event Calculus dialect that we assume in this thesis. Recall also from Section 2.3.1 that a training example is a Herbrand interpretation consisting of anything known true at *at least* two consecutive time points T and $T{+}1$ and that *covers*(T, e) means that the program $T \cup e$ has at least one stable model.

Definition 3.1 (Incremental Learning)**.** Given:

- An initially empty database of examples $\mathcal{E}$, called historical memory, in which examples that arrive over time are stored.
- A set of mode declarations M.
- A hypothesis $H_n \in \mathcal{L}(M)$ such that *covers*$(\mathsf{SDEC} \cup H_n, \mathcal{E})$ (i.e. $\mathsf{SDEC} \cup H_n$ covers each example in $\mathcal{E}$)
- A new example w_n.

Find

- A hypothesis $H_{n+1} \in \mathcal{L}(M)$ such that *covers*$(\mathsf{SDEC} \cup H_{n+1}, \mathcal{E} \cup w_n)$.

Our main challenge is to scale up ILED to a growing size of experience. To evaluate its scalability, we use a measure adopted from stream processing, where the number of passes over a dataset is often used as a measure of the efficiency of algorithms [Li and Lee, 2009; Li et al., 2004]. In this spirit, the main contribution of ILED, in addition to scaling up the main XHAIL algorithm to large data volumes, is that it adopts a "single-pass" theory revision strategy, that is, a strategy that requires at most one pass over $\mathcal{E}$ in order to compute H_{n+1} from H_n in Definition 3.1. Note that a single-pass revision strategy is far from trivial, since in principle, theory revision requires several passes over the training data [Duboc et al., 2009].

Since experience may grow over time to an extent that is impossible to maintain in the working memory, we follow an external memory approach [Biba et al., 2006a]. This implies that the learner does not have access to all past experience as a whole, but to independent sets of training data, in the form of *sliding windows*. A sliding window is a set of training examples (interpretations). The examples in a sliding window may be merged into a single example by joining the corresponding interpretations. Therefore in what follows the terms "example window" or simply "example" are used interchangeably. At time n, ILED is presented with a hypothesis H_n that accounts for the historical memory so far, and a new example window w_n. If H_n covers the new window (i.e. each example in this window) then it is returned as is, otherwise ILED starts the process of revising H_n. In this process, revision operators that retract knowledge, such as the deletion of clauses or antecedents are excluded, due to the exponential cost of backtracking in the historical memory [Badea, 2001]. The supported revision operators are thus:

- Addition of new clauses.
- Refinement of existing clauses, i.e. replacement of an existing clause with one or more specializations of that clause.

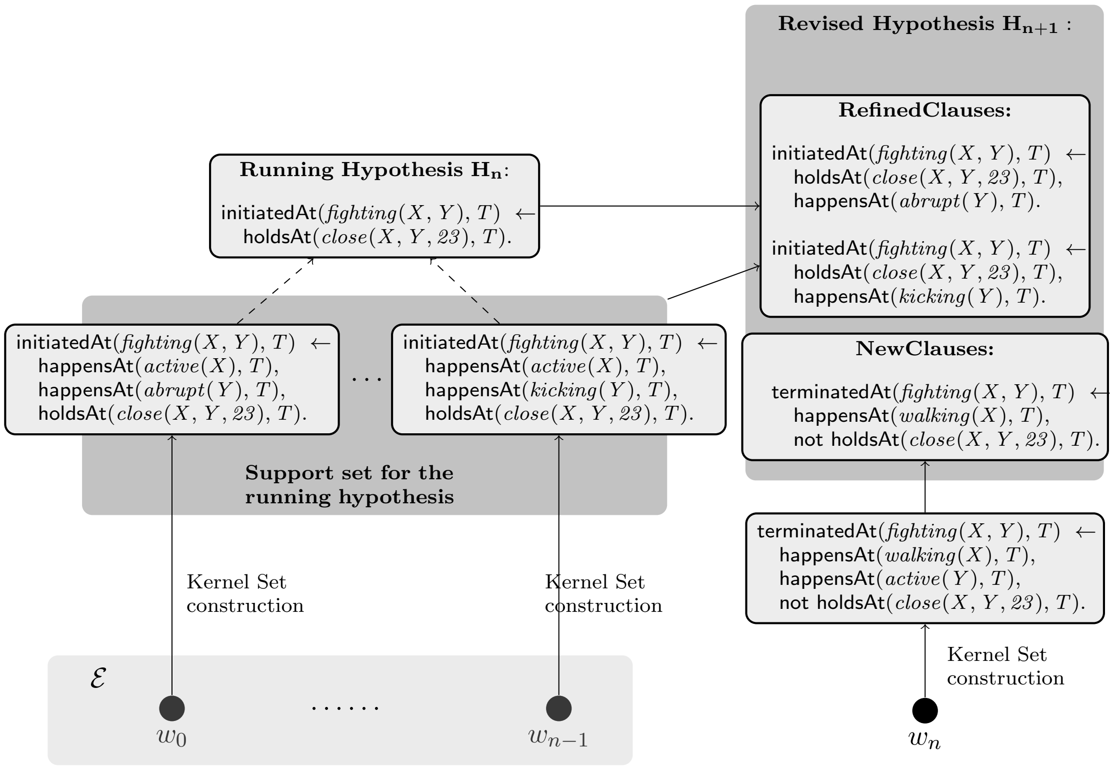
Revised Hypothesis $\mathbf{H_{n+1}}$:
RefinedClauses:
initiatedAt($fighting(X, Y), T$) ← holdsAt($close(X, Y, 23), T$), happensAt($abrupt(Y), T$).
initiatedAt($fighting(X, Y), T$) ← holdsAt($close(X, Y, 23), T$), happensAt($kicking(Y), T$).
Running Hypothesis $\mathbf{H_n}$:
initiatedAt($fighting(X, Y), T$) ← holdsAt($close(X, Y, 23), T$).
initiatedAt($fighting(X, Y), T$) ← happensAt($active(X), T$), happensAt($abrupt(Y), T$), holdsAt($close(X, Y, 23), T$).
. . .
initiatedAt($fighting(X, Y), T$) ← happensAt($active(X), T$), happensAt($kicking(Y), T$), holdsAt($close(X, Y, 23), T$).
NewClauses:
terminatedAt($fighting(X, Y), T$) ← happensAt($walking(X), T$), not holdsAt($close(X, Y, 23), T$).
Support set for the running hypothesis
terminatedAt($fighting(X, Y), T$) ← happensAt($walking(X), T$), happensAt($active(Y), T$), not holdsAt($close(X, Y, 23), T$).
Kernel Set construction
Kernel Set construction
Kernel Set construction
$\mathcal{E}$
w_0
. . . . . .
w_{n-1}
w_n

FIGURE 3.1: Revision of a hypothesis H_n in response to a new example window w_n

To treat incompleteness we add initiatedAt clauses and refine terminatedAt clauses, while to treat inconsistency we add terminatedAt clauses and refine initiatedAt clauses. The goal is to retain the *preservable* clauses of H_n intact, refine its *revisable* clauses and, if necessary, generate a set of new clauses that account for new examples in the incoming window w_n. We henceforth call a clause preservable w.r.t. a set of examples if it does not cover negatives, nor it disproves positives, and call it revisable otherwise.

Figure 3.1 illustrates the revision process with a simple example. New clauses are generated by generalizing a Kernel Set of the incoming window, as shown in Figure 3.1, where a terminatedAt/2 clause is generated from the new window w_n. To facilitate refinement of existing clauses, each clause in the running hypothesis is associated with a memory of the examples it covers throughout $\mathcal{E}$, in the form of a "bottom program", which we call *support set*. The support set is constructed gradually, from previous Kernel Sets, as new example windows arrive. It serves as a refinement search space, where the single clause in the running hypothesis H_n is refined w.r.t. the incoming window w_n into two specializations. Each such specialization is constructed by adding to the initial clause one antecedent from the two support set clauses which are presented in Figure 3.1. The revised hypothesis H_{n+1} is constructed from the refined clauses and the new ones, along with the preserved clauses of H_n, if any.

ILED's support set can be seen as the S-set in a version space [Mitchell, 1979], i.e. the space of all overly-specific hypotheses, progressively augmented as new examples arrive.

Similarly, a running hypothesis of ILED can be seen as an element of the G-set in a version space, i.e. the space of all overly-general hypotheses that account for all examples seen so far, and need to be further refined as new examples arrive.

Algorithm 2 iled(SDEC, M, H_n, w_n) **(ILED's High-Level Strategy)**
Input: *The axioms of SDEC, mode declarations M, a hypothesis* H_n *such that covers*(*SDEC*$\cup$ $H_n, \mathcal{E}$) and an example window w_n.
Output: *A hypothesis* H_{n+1} *such that covers*(*SDEC* $\cup H_{n+1}, \mathcal{E} \cup w_n$)

```
if $\neg$covers(SDEC $\cup H_n, w_n$) then
    let $K_v^{w_n}$ be a (variabilized) Kernel Set of $w_n$
    let $\langle RetainedClauses, RefinedClauses, NewClauses \rangle \leftarrow$ revise(SDEC, $H_n, K_v^{w_n}, w_n$)

    let $H' \leftarrow H_{keep} \cup RefinedClauses \cup NewClauses$
    if $NewClauses \neq \emptyset$ then
        for all $w_i \in \mathcal{E},\ 0 \leq i \leq n-1$ do
            if $\neg$covers(SDEC $\cup H', w_i$) then
                let $\langle RetainedClauses, RefinedClauses, \emptyset \rangle \leftarrow$ revise(SDEC, $H', \emptyset, w_i$)
                let $H' \leftarrow RetainedClauses \cup RefinedClauses$
    let $H_{n+1} \leftarrow H'$
else
    let $H_{n+1} \leftarrow H_n$
let $\mathcal{E} \leftarrow \mathcal{E} \cup w_n$
Return $H_{n+1}$
```

There are two key features of ILED that contribute towards its scalability: First, re-processing of past experience is necessary only in the case where new clauses are generated by a revision, and is redundant in the case where a revision consists of refinements of existing clauses only. Second, re-processing of past experience requires a single pass over the historical memory, meaning that it suffices to re-visit each past window exactly once to ensure that the output revised hypothesis H_{n+1} is complete & consistent w.r.t. the entire historical memory. These properties of ILED are due to the support set, which we next present in detail. A proof of soundness and the single-pass revision strategy of ILED is given in Proposition 3.5 at the end of this section. The Pseudocode of ILED's strategy is presented in Algorithm 2.

3.2.1 Support Set

Given a set of mode declarations M, a clause C in the mode language $\mathcal{L}(M)$ is most-specific if it does not θ-subsume any other clause in $\mathcal{L}(M)$. Intuitively, the support set of a clause C is a "bottom program" that consists of most-specific versions of the clauses that disjunctively define the concept captured by C. A formal account is given in Definition 3.2.

Definition 3.2 (Support Set). Let $\mathcal{E}$ be the historical memory, M a set of mode declarations, $\mathcal{L}(M)$ the corresponding mode language of M and $C \in \mathcal{L}(M)$ a clause. Also, let us denote by $cov_{\mathcal{E}}(C)$ the coverage of clause C in the historical memory, i.e. $cov_{\mathcal{E}}(C) = \{e \in \mathcal{E} \mid covers(\mathsf{SDEC} \cup C, e)\}$. The support set $C.supp$ of clause C is defined as follows:

$$C.supp = \bigcup_{e \in cov_{\mathcal{E}}(C)} \{D \in \mathcal{L}(M) \mid e \in cov_{\mathcal{E}}(D) \text{ and } C \preceq D \text{ and } \\ \forall D' \in \mathcal{L}(M), \text{ if } e \in cov_{\mathcal{E}}(D') \text{ then } D' \preceq D\}$$

The support set of clause C is thus defined as the set consisting of one bottom clause per each example $e \in cov_{\mathcal{E}}(C)$, i.e. one most-specific clause D of $\mathcal{L}(M)$ such that $C \preceq D$ and $covers(\mathsf{SDEC} \cup D, e)$. Assuming no length bounds on hypothesized clauses, each such bottom clause is unique[1] and covers at least one example from $cov_{E}(C)$; note that since the bottom clauses for a set of examples in $cov_{\mathcal{E}}(C)$ may coincide (i.e. be θ-subsumption equivalent – they θ-subsume each other), a clause D in $C.supp$ may cover more than one example from $cov_{\mathcal{E}}(C)$. Proposition 3.3 highlights the main property of the structure.

Proposition 3.3. *Let C be a clause in $\mathcal{L}(M)$. $C.supp$ is the most specific program of $\mathcal{L}(M)$ such that $cov_{\mathcal{E}}(C.supp) = cov_{\mathcal{E}}(C)$.*

Proof We first show that $cov_{\mathcal{E}}(C.supp) = cov_{\mathcal{E}}(C)$. For the inclusion $cov_{\mathcal{E}}(C.supp) \subseteq cov_{\mathcal{E}}(C)$, assume that $e \in cov_{\mathcal{E}}(C.supp)$, i.e. e is covered by a $D \in C.supp$. But C θ-subsumes D, therefore $e \in cov_{\mathcal{E}}(C)$. For the inverse inclusion, assume that $e \in cov_{\mathcal{E}}(C)$ and let D be the most-specific clause of $\mathcal{L}(M)$, such that $e \in cov_{\mathcal{E}}(D)$ $(\star)$ (observe that if no such D exists, with $D \neq C$, then C itself is the most-specific clause with the required property). Then by definition, $D \in C.supp$ and from $(\star)$ above we have that $e \in cov_{\mathcal{E}}(C.supp)$, establishing the inclusion $cov_{\mathcal{E}}(C) \subseteq cov_{\mathcal{E}}(C.supp)$.

The fact that $C.supp$ is the most-specific program of $\mathcal{L}(M)$ with this property follows immediately from Definition 3.2, since each clause in $\mathcal{L}_i(C.supp)$ is most-specific in $\mathcal{L}(M)$ with the property of covering at least one example from $cov_{\mathcal{E}}(C)$.

■

Proposition 3.3 implies that clause C and its support set $C.supp$ define a space $\mathcal{S}$ of specializations of C, each of which is bound by a most-specific specialization, among

[1]The bottom clause relative to an example can be large, or even infinite. To constrain its size, several restrictions are imposed on the language, such as a maximum clause length, or a maximum variable depth. We refrain from assuming extra language bias related to clause length and instead, for the purposes of this work, we assume a finite domain and impose no particular bounds on clause length. In such context, the bottom clause of an example e is unique and results from the *ground* most-specific clause that covers e, by properly replacing terms with variables, as indicated by the mode declarations.

those that cover the positive examples that C covers. In other words, for every $D \in \mathcal{S}$ there is a $C_s \in C.supp$ so that $C \preceq D \preceq C_s$ and C_s covers at least one example from $cov_{\mathcal{E}}(C)$. Moreover, Proposition 3.3 ensures that space $\mathcal{S}$ contains refinements of clause C that collectively preserve the coverage of C in the historical memory. The purpose of $C.supp$ is thus to serve as a search space for refinements R_C of clause C for which $C \preceq R_C \preceq C.supp$ holds. Since such refinements preserve C's coverage of positive examples, clause C may be refined w.r.t. a window w_n, avoiding the overhead of re-testing the refined program on $\mathcal{E}$ for completeness. However, to ensure that the support set can indeed be used as a refinement search space, one must ensure that $C.supp$ will always contain such a refinement R_C. The proof is given in Proposition 3.4.

Proposition 3.4. *Let $H_n \in \mathcal{L}(M)$ be as in the Incremental Learning setting (Definition 3.1), i.e. $covers(\mathit{SDEC} \cup H_n, \mathcal{E})$, and w_n be an example window. Assume also that there exists a hypothesis $H_{n+1} \in \mathcal{L}(M)$, such that $covers(\mathit{SDEC} \cup H_{n+1}, \mathcal{E} \cup w_n)$, and that a clause $C \in H_n$ is revisable w.r.t. window w_n. Then $C.supp$ contains a refinement R_C of C, which is preservable w.r.t. w_n.*

Proof Assume, towards contradiction, that each refinement R_C of C, contained in $C.supp$ is revisable w.r.t. w_n. It then follows that $C.supp$ itself is revisable w.r.t. w_n, i.e. it either covers some negative examples, or it disproves some positive examples in w_n. Let $e_1 \in w_n$ be such an example that $C.supp$ fails to satisfy, and assume for simplicity that a single clause $C_s \in C.supp$ is responsible for that. By definition, C_s covers at least one positive example e_2 from $\mathcal{E}$ and furthermore, it is a most-specific clause, within $\mathcal{L}_i(M)$, with that property. It then follows that e_1 and e_2 cannot both be accounted for, under the given language bias $\mathcal{L}(M)$, i.e. there exists no hypothesis $H_{n+1} \in \mathcal{L}(M)$ such that $covers(\mathrm{SDEC} \cup H_{n+1}, \mathcal{E} \cup w_n)$, which contradicts our assumption. Hence $C.supp$ is preservable w.r.t. w_n and it thus contains a refinement R_C of C, which is preservable w.r.t. w_n. ■

The construction of the support set, presented in Algorithm 3, is a process that starts when C is added in the running hypothesis and continues as long as new example windows arrive. While this happens, clause C may be refined or retained, and its support set is updated accordingly. The details of Algorithm 3 are presented in Example 3.1, which also demonstrates how ILED processes incoming examples and revises hypotheses.

Example 3.1. *Consider the annotated examples and running hypothesis related to the* fighting *high-level event from the activity recognition application shown in Table 3.1. We assume that ILED starts with an empty hypothesis and an empty historical memory, and that w_1 is the first input example window. The currently empty hypothesis does not cover the provided examples, since in w_1* fighting *between persons id_1 and id_2 is initiated at time 10 and thus holds at time 11. Hence ILED starts the process of generating an initial hypothesis. In the case of an empty hypothesis, ILED reduces to XHAIL and operates on a Kernel Set of w_1 only. The variabilized Kernel Set in this case will be the single-clause program K_1 presented*

Algorithm 3 Support set construction and maintenance

```
1: let $w_n \notin \mathcal{E}$ be an example window, $H_n$ a current hypothesis and
   $H'_n = NewClauses \cup RefinedClauses \cup RetainedClauses$ a revision of $H_n$, generated
   in $w_n$.
2: for all $C \in H'_n$ do
3:     if $C \in NewClauses$ then
4:         $C.supp \leftarrow \{D \in K \mid C \preceq D\}$, where $K$ is the variabilized Kernel Set of $w_n$
           from which $NewClauses$ is generated.
5:     else if $C \in RefinedClauses$ then
6:         $C.supp \leftarrow \{D \in C_{parent}.supp \mid C \preceq D\}$, where $C_{parent}$ is the "ancestor"
           clause of $C$, i.e. the clause from which $C$ results by specialization.
7:     else
8:         let $e_C^{w_n}$ be the true positives that $C$ covers in $w_n$, if $C$ is an initiatedAt clause, or
               the true negatives that $C$ covers, if it is a terminatedAt clause.
9:         if SDEC $\cup\ C.supp \nvDash e_C^{w_n}$ then
10:            let $K$ be a variabilized Kernel Set of $w_n$.
11:            $C.supp \leftarrow C.supp \cup K'$, where $K' \subseteq K$, such that SDEC $\cup\ K' \vDash e_C^{w_n}$
```

in Table 3.1, generated from the corresponding ground clause. Generalizing this Kernel Set yields a minimal hypothesis that covers w_1. One such hypothesis is clause C shown in Table 3.1. ILED stores w_1 in $\mathcal{E}$ and initializes the support set of the newly generated clause C as in line 3 of Algorithm 3, by selecting from K_1 the clauses that are θ-subsumed by C, in this case, K_1's single clause.

Window w_2 arrives next. In w_2, fighting *is initiated at time 20 and thus holds at time 21. The running hypothesis correctly accounts for that and thus no revision is required. However, $C.supp$ does not cover w_2 and unless proper actions are taken, property (i) of Proposition 3.3 will not hold once w_2 is stored in $\mathcal{E}$. ILED thus generates a new Kernel Set K_2 from window w_2, as presented in Table 3.1, and updates $C.supp$ as shown in lines 7-11 of Algorithm 3. Since C θ-subsumes K_2, the latter is added to $C.supp$, which now becomes $C.supp = \{K_1, K_2\}$. Now $cov_{\mathcal{E}}(C.supp) = cov_{\mathcal{E}}(C)$, hence in effect, $C.supp$ is a summarization of the coverage of clause C in the historical memory.*

Window w_3 arrives next, which has no positive examples for the initiation of fighting. *The running hypothesis is revisable in window w_3, since clause C covers a negative example at time 31, by means of initiating the fluent fighting(id_1, id_2) at time 30. To address the issue, ILED searches $C.supp$, which now serves as a refinement search space, to find a refinement R_C that rejects the negative example, and moreover $R_C \preceq C.supp$. Several choices exist for that. For instance, the following program*

initiatedAt$(fighting(X,Y),T) \leftarrow$
  happensAt$(active(X),T)$,
  happensAt$(abrupt(Y),T)$.

initiatedAt$(fighting(X,Y),T) \leftarrow$
  happensAt$(active(X),T)$,
  happensAt$(kicking(Y),T)$.

is such a refinement R_C, since it does not cover the negative example in w_3 and subsumes $C.supp$. ILED however is biased towards minimal theories, in terms of the overall number

Window w_1	
Narrative happensAt($active(id_1), 10$). happensAt($abrupt(id_2), 10$). holdsAt($close(id_1, id_2, 23), 10$).	**Annotation** not holdsAt($fighting(id_1, id_2), 10$). holdsAt($fighting(id_1, id_2), 11$).
Kernel Set initiatedAt($fighting(id_1, id_2), 10) \leftarrow$ happensAt($active(id_1), 10$), happensAt($abrupt(id_2), 10$) holdsAt($close(id_1, id_2, 23), 10$)	**Variabilized Kernel Set** $K_1 =$ initiatedAt($fighting(X, Y), T) \leftarrow$ happensAt($active(X), T$), happensAt($abrupt(Y), T$), holdsAt($close(X, Y, 23), T$).
Running Hypothesis $C =$ initiatedAt($fighting(X, Y), T) \leftarrow$ happensAt($active(X), T$).	**Support Set** $C.supp = \{K_1\}$
Window w_2	
Narrative happensAt($active(id_1), 20$). happensAt($kicking(id_2), 20$). holdsAt($close(id_1, id_2, 23), 20$).	**Annotation** not holdsAt($fighting(id_1, id_2), 20$). holdsAt($fighting(id_1, id_2), 21$).
Kernel Set initiatedAt($fighting(id_1, id_2), 20) \leftarrow$ happensAt($active(id_1), 20$), happensAt($kicking(id_2), 20$) holdsAt($close(id_1, id_2, 23), 20$)	**Variabilized Kernel Set** $K_2 =$ initiatedAt($fighting(X, Y), T) \leftarrow$ happensAt($active(X), T$), happensAt($kicking(Y), T$), holdsAt($close(X, Y, 23), T$).
Running Hypothesis Remains unchanged	**Support Set** $C.supp = \{K_1, K_2\}$
Window w_3	
Narrative happensAt($active(id_1), 30$). happensAt($walking(id_2), 30$). not holdsAt($close(id_1, id_2, 23), 30$).	**Annotation** not holdsAt($fighting(id_1, id_2), 30$). not holdsAt($fighting(id_1, id_2), 31$).
Revised Hypothesis $C_1 =$ initiatedAt($fighting(X, Y), T) \leftarrow$ happensAt($active(X), T$), holdsAt($close(X, Y, 23), T$).	**Support Set** $C_1.supp = \{K_1, K_2\}$

TABLE 3.1: Knowledge for Example 3.1

of literals and would prefer the more compressed refinement C_1, shown in Table 3.1, which also rejects the negative example in w_3 and subsumes $C.supp$. Clause C_1 replaces the initial clause C in the running hypothesis. The hypothesis now becomes complete and consistent w.r.t. $\mathcal{E}$. Note that the hypothesis was refined by local reasoning only, i.e. reasoning within window w_3 and the support set, avoiding costly look-back in the historical memory. The support set of the new clause C_1 is initialized (line 5 of Algorithm 3), by selecting the subset of the support set of its parent clause that is θ-subsumed by C_1. In this case $C_1 \preceq C.supp = \{K_1, K_2\}$, hence $C_1.supp = C.supp$.

The support set of a clause C is a compressed enumeration of the examples that C covers throughout the historical memory. It is compressed because each variabilized clause in the set is expected to encode many examples. In contrast, a ground version of the support set would be a plain enumeration of examples, since in the general case, it would require one ground clause per example. The main advantage of the "lifted" character of the support set over a plain enumeration of the examples is that it requires much less memory to encode the necessary information, an important feature in large-scale (temporal) applications. Moreover, given that training examples are typically characterized by heavy repetition, abstracting away redundant parts of the search space results in a memory structure that is expected to grow in size slowly, allowing for fast search that scales to a large amount of historical data.

We conclude this section with the proof of soundness for ILED and deriving an upper bound on the number of passes over the data that are required to learn/revise a hypothesis.

Proposition 3.5 (Soundness and Single-pass Theory Revision). *Assume the incremental learning setting described in Definition 3.1. ILED requires at most one pass over $\mathcal{E}$ to compute H_{n+1} from H_n.*

Proof For simplicity and without loss of generality, we assume that when a new example window w_n arrives, ILED revises H_n by a single-clause revision, i.e. (a) by specializing a single clause $C \in H_n$ or (b) adding a new clause C'. The case of multi-clause revisions, i.e. either specializations of multiple clauses or additions of such follows from the basic argument for the single clause case.

In case (a), clause C is replaced by a refinement R_C such that $C \preceq R_C \preceq C.supp$. By property (iii) of the support set (see Proposition 3.3), R_C covers all positive examples that C covers in $\mathcal{E}$, hence for the hypothesis $H_{n+1} = (H_n \setminus C) \cup R_C$ it holds that $covers(\mathsf{SDEC} \cup H_{n+1}, \mathcal{E})$ and furthermore $covers(\mathsf{SDEC} \cup H_{n+1}, w_n)$. Hence $covers(\mathsf{SDEC} \cup H_{n+1}, \mathcal{E} \cup w_n)$, from which soundness for H_{n+1} follows. In this case, H_{n+1} is constructed from H_n in a single step, i.e. by reasoning within w_n without re-seeing other windows from $\mathcal{E}$.

In case (b), H_n is revised w.r.t. w_n to a hypothesis $H'_n = H_n \cup C'$, where C' is a new clause that results from the generalization of a Kernel Set of w_n. In response to the new clause addition, each window in $\mathcal{E}$ must be checked and C' must be refined if necessary. Let $\mathcal{E}_{tested}$ denote the fragment of $\mathcal{E}$ that has been tested at each point in time. Initially, i.e. once C' is generated from w_n, it holds that $\mathcal{E}_{tested} = w_n$. At each window that is tested, clause C' may (i) remain intact, (ii) be refined, or (iii) one of its refinements may be further refined. Assume that $w_k,\ k < n$ is the first window where the new clause C' must be refined. At this point, $\mathcal{E}_{tested} = \{w_i \in \mathcal{E} \mid k < i \leq n\}$, and it holds that C' is preservable in $\mathcal{E}_{tested}$, since C' has not yet been refined. In w_k, clause C' is replaced by a refinement $R_{C'}$ such that $C' \preceq R_{C'} \preceq C'.supp$. $R_{C'}$ is preservable in $\mathcal{E}_{tested}$, since it is a refinement of a preservable clause, and furthermore, it covers all positive examples that C' covers in w_n, by means of the properties of the support set. Hence the hypothesis $H''_n = (H'_n \smallsetminus C') \cup R_{C'}$ is complete & consistent w.r.t. $\mathcal{E}_{tested}$. The same argument shows that if $R_{C'}$ is further refined later on (case (iii) above), the resulting hypothesis remains complete and consistent w.r.t. $\mathcal{E}_{tested}$. Hence, when all windows have been tested, i.e. when $\mathcal{E}_{tested} = \mathcal{E}$, the resulting hypothesis H_{n+1} is complete & consistent w.r.t. $\mathcal{E} \cup w_n$ and furthermore, each window in $\mathcal{E}$ has been re-seen exactly once, thus H_{n+1} is computed with a single pass over $\mathcal{E}$. ■

Proposition 3.5 refers to a setting where examples arrive over time and a hypothesis is revised w.r.t. each such example. ILED may also be used in a stationary setting, where the entire training set resides in disk. In this case it can "simulate" the incremental setting by splitting the training data into chunks (a chunk in this case is simply a window, as defined in this chapter) and processing one such chunk at a time. Based on Proposition 3.5, it can be shown that in this setting ILED needs two passes over the training data to learn a sound hypothesis from scratch. That is, if n is the number of data chunks, ILED sees each data chunk exactly twice and therefore it needs $2n$ passes over the data. This is shown in Proposition 3.6.

Proposition 3.6 (Extending Proposition 3.5 to the Stationary Setting). *Let $\mathcal{D}$ be a stationary dataset and $\mathcal{D}_n$ a partition of $\mathcal{D}$ into n example windows (chunks). To learn a sound hypothesis, ILED needs to see each window twice, therefore it needs $2n$ passes over the data.*

Proof The argument is based on the following strategy that "simulates" the incremental learning setting where examples arrive over time: In a first pass over the training set, ILED uses its core functionality to generate a hypothesis H_1 that covers the entirety of positive examples in $\mathcal{D}$, without taking into account the negative examples in each window. This can be done in a single pass over $\mathcal{D}$ (seeing each window once), by simply generating new Kernel Sets from each window with uncovered positives and generalizing these Kernels Sets as much as needed, respecting ILED's minimality bias. Each set of clauses that is obtained from each window is added to H_1 (which is initially empty), while the support set of each such clause is properly populated, in the regular way, as described earlier in this section. What remains in order to obtain a sound hypothesis is to properly specialize the clauses in hypothesis H_1 so that they do not cover any negative examples. But by means of the properties of the support set and the argument in Proposition 3.5, it follows that a single additional pass over $\mathcal{D}$ suffices to do that: In each step of that pass, all clauses that are inconsistent w.r.t. a window are specialized to exclude the negative examples they cover in that window, while they preserve all the positive examples in $\mathcal{D}_{tested}$, the part of $\mathcal{D}$ that has already been checked at that particular step. Therefore, ILED needs two passes in total to learn a sound hypothesis. ■

Algorithm 4 revise(SDEC, $H_n, w_n, K_v^{w_n}$)
Input: *The axioms of* SDEC*, a running hypothesis* H_n *an example window* w_n *and a variabilized Kernel Set* $K_v^{w_n}$ *of* w_n*.*
Output: *A revised hypothesis* H'_n

```
1: let $U(K_v^{w_n}, H_n) \leftarrow$ GeneralizationTransformation($K_v^{w_n}$)$\cup$
                              RefinementTransformation($H_n$)
2: let $\Phi$ be the abductive task $\Phi = ALP(\mathsf{SDEC} \cup U(K_v^{w_n}, H_n), \{use/2, use/3\}, w_n)$
3: if $\Phi$ has a solution then
4:        let $\Delta$ be a minimal solution of $\Phi$
5:        let $NewClauses = \{\alpha_i \leftarrow \delta_i^1 \wedge \ldots \wedge \delta_i^n \mid$
                   $\alpha_i$ is the head of the $i-$th clause $C_i \in K_v^{w_n}$
                   and $\delta_i^j$ is the $j-$th body literal of $C_i$
                   and $use(i,0) \in \Delta$ and $use(i,j) \in \Delta, 1 \leq j \leq n$ }
6:        let $RefinedClauses = \{ head(C_i) \leftarrow body(C_i) \wedge \delta_i^{j,k_1} \wedge \ldots \wedge \delta_i^{j,k_m} \mid$
                   $C_i \in H_n$ and $use(i,j,k_l) \in \Delta$, where
                   $1 \leq l \leq m, 1 \leq j \leq |C_i.supp|$ }
7:     let $RetainedClauses = \{C_i \in H_n \mid use(i,j,k) \notin \Delta$ for any $j,k\}$
8:     let $RefinedClauses$ = ReduceRefined($NewClauses, RefinedClauses, RetainedClauses$)
9: else
10:    Return No Solution
11: Return $\langle RetainedClauses, RefinedClauses, NewClauses \rangle$
```

3.2.2 Implementing Revisions

Algorithm 4 presents the revision function of ILED. The input consists of SDEC as background knowledge, a running hypothesis H_n, an example window w_n and a variabilized Kernel Set $K_v^{w_n}$ of w_n. The clauses of $K_v^{w_n}$ and H_n are subject to the GeneralizationTansformation and the RefinementTransformation respectively, presented in Table 3.2. The former is the transformation discussed in Section 2.4.3, that turns the Kernel Set into a defeasible program, allowing the construction of new clauses. The RefinementTransformation aims at the refinement of the clauses of H_n using their support sets. It involves two fresh predicates, $exception/3$ and $use/3$. For each clause $D_i \in H_n$ and for each of its support set clauses $\Gamma_i^j \in D_i.supp$, one new clause:

$$head(D_i) \leftarrow body(D_i) \wedge \mathsf{not}\ exception(i, j, v(head(D_i)))$$

is generated, where $v(head(D_i))$ is a term that contains the variables of $head(C_i)$. Then an additional clause $exception(i, j, v(head(D_i))) \leftarrow use(i, j, k) \wedge \mathsf{not}\ \delta_i^{j,k}$ is generated, for each body literal $\delta_i^{j,k} \in \Gamma_i^j$.

The syntactically transformed clauses are put together in a program $U(K_v^{w_n}, H_n)$ (line 1 of Algorithm 4), which is used as a background theory along with SDEC. A minimal set of $use/2$ and $use/3$ atoms is abduced as a solution to the abductive task Φ in line 2 of Algorithm 4. Abduced $use/2$ atoms are used to construct a set of

GeneralizationTransformation	RefinementTransformation
Input: A variabilized Kernel set K_v	**Input:** A running hypothesis H_n
For each clause $D_i = \alpha_i \leftarrow \delta_i^1, \dots, \delta_i^n \in F_v$: Add an extra atom $use(i, 0)$ to the body of D_i and replace each body literal δ_i^j with a new atom of the form $try(i, j, v(\delta_i^j))$, where $v(\delta_i^j)$ contains the variables that appear in δ_i^j. Generate two new clauses of the form $try(i, j, v(\delta_i^j)) \leftarrow use(i, j), \delta_i^j$ and $try(i, j, v(\delta_i^j)) \leftarrow \mathsf{not}\ use(i, j)$ for each δ_i^j.	**For each** clause $D_i \in H_n$: **For each** clause $\Gamma_i^j \in D_i.supp$ Generate one clause $\alpha_i \leftarrow body(D_i) \wedge \mathsf{not}\ exception(i, j, v(\alpha_i))$ where α_i is the head of D_i and $v(\alpha_i)$ contains its variables. Generate one clause $exception(i, j, v(a_i)) \leftarrow use(i, j, k), \mathsf{not}\ \delta_i^{j,k}$ for each body literal $\delta_i^{j,k}$ of Γ_i^j.

TABLE 3.2: Syntactic transformations performed by ILED.

NewClauses, as discussed in Section 2.4.3 (line 5 of Algorithm 4). These new clauses account for some of the examples in w_n, which cannot be covered by existing clauses in H_n. The abduced $use/3$ atoms indicate clauses of H_n that must be refined. From these atoms, a refinement R_{D_i} is generated for each incorrect clause $D_i \in H_n$, such that $D_i \preceq R_{D_i} \preceq D_i.supp$ (line 6 of Algorithm 4). Clauses that lack a corresponding $use/3$ atom in the abductive solution are retained (line 7 of Algorithm 4).

The intuition behind refinement generation is as follows: Assume that clause $D_i \in H_n$ must be refined. This can be achieved by means of the extra clauses generated by the RefinementTransformation. These clauses provide definitions for the exception atom, namely one for each body literal in each clause of $D_i.supp$. From these clauses, one can satisfy the exception atom by satisfying the complement of the corresponding support set literal and abducing the accompanying $use/3$ atom. Since an abductive solution Δ is minimal, the abduced $use/3$ atoms correspond precisely to the clauses that must be refined.

Hence, each inconsistent clause $D_i \in H_n$ and each $\Gamma_i^j \in D_i.supp$ correspond to a set of abduced $use/3$ atoms of the form $use(i, j, k_1), \dots, use(i, j, k_n)$. These atoms indicate that a specialization of D_i may be generated by adding to the body of D_i the literals $\delta_i^{j,k_1}, \dots, \delta_i^{j,k_n}$ from Γ_i^j. Then a refinement R_{D_i} such that $D_i \preceq R_{D_i} \preceq D_i.supp$ may be generated by selecting one specialization of clause D_i from each support set clause in $D_i.supp$.

Example 3.2. *Table 3.3 presents the process of ILED's refinement. The annotation lacks positive examples and the running hypothesis consists of a single clause C, with a support set of two clauses. Clause C is inconsistent since it entails two negative examples, namely* holdsAt*($fighting(id_1, id_2)$, 2) and* holdsAt*($fighting(id_3, id_4)$, 3). The program that results by applying the RefinementTransformation to the support set of clause C is presented in Table 3.3, along with a minimal abductive explanation of the examples, in terms of $use/3$ atoms. Atoms $use(1, 1, 2)$ and $use(1, 1, 3)$ correspond respectively to the second and third body*

Input	
Narrative	**Annotation**
happensAt$(abrupt(id_1),\ 1)$.	not holdsAt$(fighting(id_1, id_2), 1)$.
happensAt$(inactive(id_2),\ 1)$.	not holdsAt$(fighting(id_3, id_4), 1)$.
holdsAt$(close(id_1, id_2, 23),\ 1)$.	not holdsAt$(fighting(id_1, id_2), 2)$.
happensAt$(abrupt(id_3),\ 2)$.	not holdsAt$(fighting(id_3, id_4), 2)$.
happensAt$(abrupt(id_4),\ 2)$.	not holdsAt$(fighting(id_1, id_2), 3)$.
not holdsAt$(close(id_3, id_4, 23),\ 2)$.	not holdsAt$(fighting(id_3, id_4), 3)$.
Running hypothesis	**Support set**
$C =$ initiatedAt$(fighting(X,\ Y),\ T) \leftarrow$	$C_s^1 =$ initiatedAt$(fighting(X,\ Y),\ T) \leftarrow$
happensAt$(abrupt(X),\ T)$.	happensAt$(abrupt(X),\ T)$,
	happensAt$(abrupt(Y),\ T)$,
	holdsAt$(close(X,\ Y, 23),\ T)$.
	$C_s^2 =$ initiatedAt$(fighting(X,\ Y),\ T) \leftarrow$
	happensAt$(abrupt(X),\ T)$,
	happensAt$(active(Y),\ T)$,
	holdsAt$(close(X,\ Y, 23),\ T)$.
Refinement transformation:	
From C_s^1 :	**From** C_s^2 :
initiatedAt$(fighting(X, Y),\ T) \leftarrow$	initiatedAt$(fighting(X, Y),\ T) \leftarrow$
happensAt$(abrupt(X),\ T)$,	happensAt$(abrupt(X),\ T)$,
not $exception(1, 1, vars(X, Y, T))$.	not $exception(1, 2, vars(X, Y, T))$.
$exception(1, 1, vars(X, Y, T)) \leftarrow$	$exception(1, 2, vars(X, Y, T)) \leftarrow$
$use(1, 1, 2)$, not happensAt$(abrupt(Y), T)$.	$use(1, 2, 2)$, not happensAt$(active(Y), T)$.
$exception(1, 1, vars(X, Y, T)) \leftarrow$	$exception(1, 2, vars(X, Y, T)) \leftarrow$
$use(1, 1, 3)$, not holdsAt$(close(X, Y, 23), T)$.	$use(1, 2, 3)$, not holdsAt$(close(X, Y, 23), T)$.
Minimal abductive solution	**Generated refinements**
$\Delta = \{use(1, 1, 2), use(1, 1, 3), use(1, 2, 2)\}$	initiatedAt$(fighting(X,\ Y),\ T) \leftarrow$
	happensAt$(abrupt(X),\ T)$,
	happensAt$(abrupt(Y),\ T)$,
	holdsAt$(close(X,\ Y, 23),\ T)$.
	initiatedAt$(fighting(X,\ Y),\ T) \leftarrow$
	happensAt$(abrupt(X),\ T)$,
	happensAt$(active(Y),\ T)$.

TABLE 3.3: Clause refinement by ILED.

literals of the first support set clause, which are added to the body of clause C, resulting in the first specialization presented in Table 3.3. The third abduced atom $use(1, 2, 2)$ *corresponds to the second body literal of the second support set clause, which results in the second specialization in Table 3.3. Together, these specializations form a refinement of clause C that subsumes* $C.supp$.

Minimal abductive solutions imply that the running hypothesis is minimally revised. Revisions are minimal w.r.t. the length of the clauses in the revised hypothesis, but are not minimal w.r.t. the number of clauses, since the refinement strategy described above may result in refinements that include redundant clauses: Selecting one specialization from each support set clause to generate a refinement of a clause is sub-optimal, since there may exist other refinements with fewer clauses that also subsume the whole support set, as Example 3.1 demonstrates. To avoid unnecessary increase of the hypothesis size, the generation of refinements is followed by a "reduction" step (line 8 of Algorithm 4). The ReduceRefined function works as follows. For each refined clause C, it first generates all possible refinements from $C.supp$. This can be realized with the abductive refinement technique described above. The only difference is that the abductive solver is instructed to find all abductive explanations in terms of $use/3$ atoms, instead of just a single explanation. Once all refinements are generated, ReduceRefined searches the revised hypothesis, augmented with all refinements of clause C, to find a reduced set of refinements of C that subsume $C.supp$.

3.3 Discussion and Related Work

Like XHAIL, ILED aims for soundness, that is, hypotheses which cover all given examples. XHAIL ensures soundness by generalizing all examples in one go. In contrast, ILED has access to a memory of past experience for which newly acquired knowledge must account. Concerning completeness, XHAIL is a state-of-the-art system among its Inverse Entailment-based peers [Corapi et al., 2010; Ray, 2009a]. Although ILED preserves XHAIL's soundness, it does not preserve its completeness properties, due to the fact that ILED operates incrementally to gain efficiency. Thus there are cases where a hypothesis can be discovered by XHAIL, but be missed by ILED. As an example, consider cases where a target hypothesis captures long-term temporal relations in the data, as for instance, in the following clause:

$$\begin{aligned}&\mathsf{initiatedAt}(moving(X,Y),T) \leftarrow\\ &\quad \mathsf{happensAt}(walking(Y),T1),\\ &\quad T1 < T.\end{aligned}$$

In such cases, if the parts of the data that are connected via a long-range temporal relation are given in different windows, ILED has no way to correlate these parts in order to discover the temporal relation. However, one can always achieve XHAIL's functionality by increasing appropriately ILED's window size.

An additional trade-off for efficiency is that not all of ILED's revisions are fully evaluated on the historical memory. For instance, selecting a particular clause in order to cover a new example, may result in a large number of refinements and an unnecessarily lengthy hypothesis, as compared to one that could have been obtained by selecting a different initial clause. On the other hand, fully evaluating all possible choices over $\mathcal{E}$ requires extensive inference. Thus simplicity and compression of hypotheses in ILED have been sacrificed for efficiency.

In ILED, a large part of the theorem proving effort that is involved in clause refinement reduces to computing subsumption between clauses, which is a hard task. Moreover, just as the historical memory grows over time, so do (in the general case) the support sets of the clauses in the running hypothesis, increasing the cost of computing subsumption. However, as in principle the largest part of a search space is redundant and the support set focuses only on its interesting parts, one would not expect that the support set will grow to a size that makes subsumption computation less efficient than inference over the entire $\mathcal{E}$. In addition, a number of optimization techniques have been developed over the years and several generic subsumption engines have been proposed [Kuzelka and Zelezny, 2008; Maloberti and Sebag, 2004; Santos and Muggleton, 2010], some of which are able to efficiently compute subsumption relations between clauses comprising thousands of literals and hundreds of distinct variables.

The limitations of existing approaches, similar to ours, which combine abduction with induction to handle non-monotonic learning tasks involving unobserved target predicates, have been discussed in Section 2.5.1. Similar limitations hold for most of the prominent theory revision approaches, which could be used in an incremental setting. We next discuss these limitations by reviewing the most important ILP theory revision systems and point out their weaknesses w.r.t. to learning with the Event Calculus.

FORTE [Richards and Mooney, 1995] is one of the most widely used ILP theory revision systems. Its revision strategy starts with the identification of *revision points*. FORTE's revision strategy begins by identifying a set of *revision points*, i.e. particular clauses in the hypothesis that fail and need to be revised. A generalization (resp. specialization) point is a clause that fails to derive a positive example (resp. derives a negative example). Each possible revision w.r.t. each revision point is then generated, based on a set of revision operators and these candidate revisions are sorted by "potential", i.e. the expected gain in the quality of the hypothesis, resulting by implementing a revision. The candidate revisions are then evaluated on the training set and the best revision for each revision point is selected and implemented. This process is repeated until either the revised hypothesis accounts for all the examples, or the quality of the hypothesis cannot be further improved. FORTE's basic disadvantages w.r.t. its usage with the Event Calculus theories are that it learns function-free Horn clauses and does not support non-Observational Predicate Learning. Also, FORTE cannot operate on an empty hypothesis (i.e. it cannot induce a hypothesis from scratch).

In [Duboc et al., 2009], FORTE is enhanced by PROGOL's bottom clause construction routine and mode declarations, towards a more efficient refinement operator. In order to refine a clause r, FORTE_MBC (the resulting system), uses mode declarations and inverse entailment to construct a bottom clause from a positive example covered by r, resulting in a more constrained search space and a more efficient clause refinement process. However, FORTE_MBC inherits FORTE's disadvantages mentioned above.

CLINT [De Raedt and Bruynooghe, 1994] is a revision system that relies on two revision operators: A generalization operator that adds new clauses or ground facts to the theory at hand and a specialization operator that retracts incorrect clauses from the theory. To generate new clauses, CLINT uses "starting clauses", i.e. most-specific clauses that cover a single positive example. These starting clauses are then maximally generalized to obtain a good hypothesis clause. CLINT is also an abductive-inductive learner, since it uses abduction in order to explain some examples. However, abduction and induction are independent and complementary, i.e. abduction is used only when the clauses induced so far fail to account for an example, in which case a set of ground facts that explain these examples are abduced and simply added to the theory at hand. As a result, CLINT cannot handle non-Observational Predicate Learning and cannot be used with the Event Calculus.

INTHELEX [Di Mauro et al., 2005, 2004; Esposito et al., 2004, 2000] learns/revises Datalog theories, using clause/literal addition/deletion, abduction and constant/variable unification as its revision operators.Several limitations make INTHELEX inappropriate for inducing/revising Event Calculus programs. First, the restriction of its input language to Datalog limits its applicability to richer, relational event domains. For instance, complex relations between entities cannot be easily expressed in INTHELEX. A workaround is to use a set of new function symbols to represent "flattened" predicates Rouveirol [1994]. However, as pointed out in Ray [2009a], this does not always preserve logical entailment Hirata [1999]. Second, the use of background knowledge is limited in INTHELEX, by excluding arithmetic functions. For instance auxiliary clauses that may be used for spatio-temporal reasoning during learning time. Third, although INTHELEX uses abduction for the completion of imperfect input data, it relies on Observational Predicate Learning, meaning that it is not able to reason with predicates which are not directly observable in the examples.

RUTH [Ade et al., 1994] is one of the earliest systems for theory revision in ILP. Given a revisable theory H and an "integrity theory" I, RUTH's first step is to identify whether H violates I and generate an example for each such violation. This example serves as a starting point for a revision. RUTH specializes clauses in response to covered negative examples and it adds new clauses, adds exceptions to clause bodies, or abduces missing explanations in response of uncovered positive examples. Each revision may generate new contradictions between the revised theory and the integrity theory. The process continues until the revised hypothesis is consistent with the integrity theory. RUTH's basic disadvantage w.r.t. its usage with the Event Calculus is that like INTHELEX, it operates on Datalog and like FORTE, it cannot learn a hypothesis from scratch.

In addition to the above-mentioned limitations, one thing that all these systems have in common is that they search the space of possible revisions by means of heuristics based on the generality order, i.e. they use generalization to handle incompleteness and specialization to handle inconsistency. This is a viable strategy in Horn logic, but as the example 2.4 in Section 2.4.2 demonstrates, it is not applicable in principle in a non-monotonic setting. A more recent approach to theory revision overcomes this difficulty by viewing theory revision as "non-monotonic ILP" [Corapi, 2012; Corapi et al., 2011a, 2008; Maggi et al., 2011]. Under this approach, given a revisable theory T, any off-the-self, non-monotonic ILP learner may be used as a theory revision system, learning a set of prescriptions based on a "meta-theory" T', which is constructed from T. The learnt prescriptions are subsequently transformed into syntactic modifications on T. This strategy resembles what XHAIL does in order to generalize a Kernel Set. XHAIL's generalization strategy relies on the non-monotonic semantics of Negation as Failure to form a hypothesis from the Kernel Set, by discarding its non-useful parts. This is done by constructing a meta-theory from the Kernel Set (using the predicates *use/2* and *try/3*) and abducing a set of *use/2* instances from this meta-theory, using the training examples as guiding integrity constraints that the meta-theory must satisfy. The abduced *use/2* instances serve as prescriptions for the selection of particular clauses and literals from the original theory (the Kernel Set), so that the selected parts of the Kernel Set satisfy the examples. This generic strategy is already a generalization revision operator (used to generate new clauses) and it may be extended to realize other operators, like addition/removal of literals to a clause, or removal of clauses from a hypothesis. A generic algorithm for theory revision as non-monotonic ILP that uses the XHAIL system as the underlying learner may be found in [Corapi, 2012; Corapi et al., 2008]. Designed to operate in full clausal logic thanks to the non-monotonic semantics of the underlying ILP learner, this approach to theory revision is able to learn and revise theories in a non-monotonic context, it is therefore appropriate to use with the Event Calculus.

Most of the theory revision systems mentioned above do not scale to large data volumes. The traditional theory revision systems (FORTE, CLINT, RUTH) were designed as "batch theory revisors", i.e. they assume a setting were a theory and a set of examples are given from the start and the goal is to revise the theory w.r.t. these examples. The same holds for the more recent approach to theory revision as non-monotonic ILP, which additionally inherits the poor scalability of the underlying non-monotonic learner. As mentioned at the beginning of this chapter, such learners employ theory-level search to learn whole theories from the entire training setting, in order to overcome difficulties with Negation as Failure. The only exception of a theory revision system whose scalability in a full memory approach has been studied to some extent is INTHELEX. In [Biba et al., 2006a] the authors propose an approach towards scaling-up INTHELEX by associating clauses in the theory at hand with examples they cover, via a relational schema. Thus, when a clause is refined, only the examples that were previously covered by this clause are checked. Similarly, when a clause is generalized, only the negative examples are checked again. The scalable version of INTHELEX presented in [Biba et al., 2006a] maintains alternative versions of the hypothesis at each step, allowing it to backtrack to previous states. In addition, it keeps in memory several statistics related to the examples that the system has already seen, such as the number of refinements that each example has caused, a "refinement history" of each clause, etc. However, despite INTHELEX's improved scalability as compared to other prominent theory revisors, its limitations related to its inability to perform non-Observational Predicate Learning and incorporate background knowledge, make it inappropriate for our purposes.

4 Experimental Evaluation for ILED

In this section, we present experimental results for ILED from two real-world applications: Activity recognition, using real data from the benchmark CAVIAR video surveillance dataset[1], as well as large volumes of synthetic CAVIAR data; and City Transport Management (CTM) using data from the PRONTO[2] project.

Part of our experimental evaluation aims to compare ILED with XHAIL. To achieve this aim we had to implement XHAIL, because the original implementation was not publicly available until recently [Bragaglia and Ray, 2014]. All experiments were conducted on a 3.2 GHz Linux machine with 4 GB of RAM. The algorithms were implemented in Python, using the Clingo[3] Answer Set Solver [Gebser et al., 2012] as the main reasoning component, and a Mongodb[4] NoSQL database for the historical memory of the examples. The code and datasets used in these experiments are available online[5].

4.1 Activity Recognition

In activity recognition, our goal is to learn definitions of high-level events, such as *fighting, moving* and *meeting*, from streams of low-level events like *walking, standing, active* and *abrupt*, as well as spatio-temporal knowledge. We use the benchmark CAVIAR dataset for experimentation. Details on the CAVIAR dataset can be found in [Artikis et al., 2010b].

CAVIAR contains noisy data mainly due to human errors in the annotation [Artikis et al., 2010b; List et al., 2005]. ILED cannot handle noise (it cannot learn non-sound hypotheses) and therefore for the experiments we manually selected a noise-free subset of CAVIAR. The resulting dataset consists of 1000 examples (that is, data for 1000 distinct time points) concerning the high-level events *moving*, *meeting* and *fighting*. These data, selected from different parts of the CAVIAR dataset, were combined into a continuous annotated stream of narrative atoms, with time ranging from 0 to 1000.

[1]http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/
[2]http://www.ict-pronto.org/
[3]http://potassco.sourceforge.net/
[4]http://www.mongodb.org/
[5]https://github.com/nkatzz/ILED

In addition to the real data, we generated synthetic data, based on the manually-developed CAVIAR event definitions described in [Artikis et al., 2010b]. In particular, streams of low-level events were created randomly and were then classified using the rules of [Artikis et al., 2010b]. The generated data consists of approximately 10^5 examples, which amounts to 100 MB of data.

The synthetic data is much more complex than the real CAVIAR data. This is due to two main reasons: First, the synthetic data includes significantly more initiations and terminations of a high-level event, thus much larger learning effort is required to explain it. Second, in the synthetic dataset more than one high-level event may be initiated or terminated at the same time point. This results in Kernel Sets with more clauses, which are hard to generalize simultaneously.

4.1.1 ILED vs XHAIL

The purpose of this experiment was to assess whether ILED can efficiently generate hypotheses comparable in size and predictive quality to those of XHAIL. To this end, we compared both systems on real and synthetic data using 10-fold cross validation with replacement. For the real data, 90% of randomly selected examples, from the total of 1000 were used for training, while the remaining 10% was retained for testing. At each run, the training data were presented to ILED in example windows of sizes 10, 50, 100. The data were presented in one batch to XHAIL. For the synthetic data, 1000 examples were randomly sampled at each run from the dataset for training, while the remaining data were retained for testing. Similar to the real data experiments, ILED operated on windows of sizes of 10, 50, 100 examples and XHAIL on a single batch.

Table 4.1 presents the experimental results. Training times are significantly higher for XHAIL, due to the increased complexity of generalizing Kernel Sets that account for the whole set of the presented examples at once. These Kernel Sets consisted, on average, of 30 to 35 clauses consisting on average of 16-20 literals, in the case of the real data, and 60 to 70 clauses with the same average length in the case of the synthetic data. In contrast, ILED had to deal with much smaller Kernel Sets (fewer number of clauses in a Kernel Set). The complexity of abductive search though, affects ILED as well, as the size of the input windows grows. ILED handles the learning task relatively well (in approximately 30 seconds) when the examples are presented in windows of 50 examples, but the training time increases almost 15 times if the window size is doubled.

Concerning the size of the produced hypothesis, the results show that in the case of real CAVIAR data, the hypotheses constructed by ILED are comparable in size with a hypothesis constructed by XHAIL. In the case of synthetic data, the hypotheses returned by both XHAIL and ILED were significantly more complex. Note that for ILED the hypothesis size decreases as the window size increases. This is reflected in the number of revisions that

	ILED			XHAIL
Real CAVIAR data	$G = 10$	$G = 50$	$G = 100$	$G = 900$
Training Time (sec)	34.15	23.04	286.74	1560.88
Revisions	11.2	9.1	5.2	–
Hypothesis size	17.82	17.54	17.5	15
Precision	98.713	99.767	99.971	99.973
Recall	99.789	99.845	99.988	99.992
Synthetic CAVIAR data	$G = 10$	$G = 50$	$G = 100$	$G = 1000$
Training Time (sec)	38.92	33.87	468	21429
Revisions	28.7	15.4	12.2	–
Hypothesis size	143.52	138.46	126.43	118.18
Precision	55.713	57.613	63.236	63.822
Recall	68.213	71.813	71.997	71.918

TABLE 4.1: Comparison of ILED and XHAIL. G is the window granularity.

ILED performs, which is significantly smaller when the input comes in larger batches of examples. In principle, the richer the input, the better the hypothesis that is initially acquired, and consequently, the less the need for revisions in response to new training instances. There is a trade-off between the window size (thus the complexity of the abductive search) and the number of revisions. A small number of revisions on complex data (i.e. larger windows) may have a greater total cost in terms of training time, as compared to a greater number of revisions on simpler data (i.e. smaller windows). For example, in the case of window size 100 for the real CAVIAR data, ILED performs 5 revisions on average and requires significantly more time than in the case of a window size 50, where it performs 9 revisions on average. On the other hand, training times for windows of size 50 are slightly better than those obtained when the examples are presented in smaller windows of size 10. In this case, the "unit cost" of performing revisions on a single window are comparable between windows of size 10 and 50. Thus the overall cost in terms of training time is determined by the total number of revisions, which is greater in the case of window size 10.

Concerning predictive quality, the results indicate that ILED's precision and recall scores are comparable to those of XHAIL's. For larger input windows, precision and recall are almost the same as those of XHAIL's. This is because ILED produces better hypotheses from larger input windows. Precision and recall are lower in the case of synthetic data for both systems, because the test set in this case is much larger and more complex than in the case of real data.

4.1.2 ILED **Scalability**

The purpose of this experiment was to assess the scalability of ILED. The experimental setting was as follows: Sets of examples of varying sizes were randomly sampled from the synthetic dataset. Each such example set was used as a training set in order to acquire an initial hypothesis using ILED. Then a new window which did not satisfy the

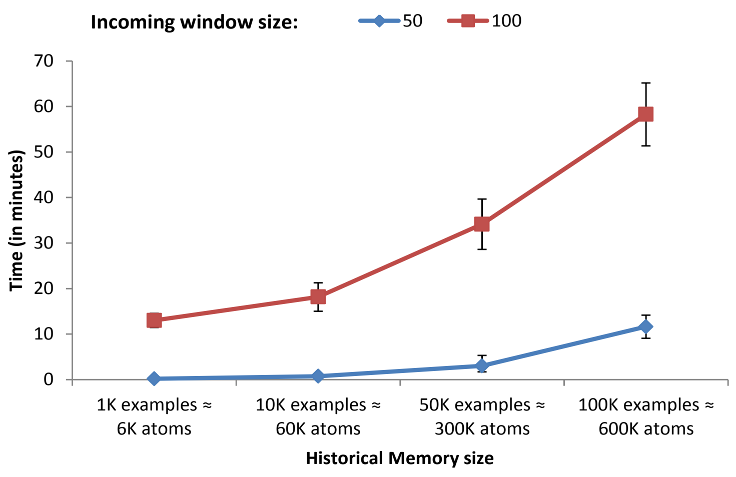

FIGURE 4.1: Average times needed for ILED to revise an initial hypothesis in the face of new evidence presented in windows of size 10, 50 and 100 examples. The initial hypothesis was obtained from a training set of varying size (1K, 10K, 50K and 100K examples) which subsequently served as the historical memory.

hypothesis at hand was randomly selected and presented to ILED, which subsequently revised the initial hypothesis in order to account for both the historical memory (the initial training set) and the new evidence. For historical memories ranging from 10^3 to 10^5 examples, a new training window of size 10, 50 and 100 was selected from the whole dataset. The process was repeated ten times for each different combination of historical memory and new window size. Figure 4.1 presents the average revision times. The revision times for window sizes of 10 and 50 examples were very close and therefore omitted to avoid clutter. The results indicate that revision time grows polynomially in the size of the historical memory.

4.2 City Transport Management

In this section we present experimental results from the domain of City Transport Management (CTM), using data from the PRONTO[6] project. In PRONTO, the goal was to inform the decision-making of transport officials by recognising high-level events related to the punctuality of a public transport vehicle (bus or tram), passenger/driver comfort and safety. These high-level events were requested by the public transport control centre of Helsinki, Finland, in order to support resource management. Low-level events were provided by sensors installed in buses and trams, reporting on changes in position, acceleration/deceleration, in-vehicle temperature, noise level and passenger density. At the time of the project, the available datasets included only a subset of the anticipated low-level event types as some low-level event detection components were not functional. Therefore, a synthetic dataset was generated. The synthetic PRONTO data has proven to be considerably more challenging for event recognition than the real data [Artikis

[6]http://www.ict-pronto.org/

et al., 2015a], and therefore we chose the former for evaluating ILED. The CTM dataset contains $5 \cdot 10^4$ examples, which amount approximately to 70 MB of data.

In contrast to the activity recognition application, the manually developed event definitions of CTM form a hierarchy. In these definitions, it is possible to define a function level that maps high-level events to non-negative integers as follows: A level-1 event is defined in terms of low-level events (input data) only. A level-n event is defined in terms of at least one level-$n-1$ event and a possibly empty set of low-level events and high-level events of level below $n-1$. Hierarchical definitions are significantly more complex to learn compared to non-hierarchical ones. This is because initiations and terminations of events in the lower levels of the hierarchy appear in the bodies of event definitions in the higher levels, hence all target definitions must be learnt simultaneously. As we show in the experiments, this has a striking effect on the learning effort. A solution for simplifying the learning task is to utilize knowledge about the domain (the hierarchy), learn event definitions separately, and use the acquired theories from lower levels of the hierarchy as non-revisable background knowledge when learning event definitions for the higher levels. A part of the CTM hierarchy is presented in Figure 4.2. Consider the following fragment:

$$\begin{aligned}&\mathsf{initiatedAt}(punctuality(Id,\ nonPunctual),\ T) \leftarrow \\ &\qquad \mathsf{happensAt}(stopEnter(Id,\ StopId,\ late),\ T).\end{aligned} \tag{4.1}$$

$$\begin{aligned}&\mathsf{initiatedAt}(punctuality(Id,\ nonPunctual),\ T) \leftarrow \\ &\qquad \mathsf{happensAt}(stopLeave(Id,\ StopId,\ early),\ T).\end{aligned} \tag{4.2}$$

$$\begin{aligned}&\mathsf{terminatedAt}(punctuality(Id,\ nonPunctual),\ T) \leftarrow \\ &\qquad \mathsf{happensAt}(stopEnter(Id,\ StopId,\ early),\ T).\end{aligned} \tag{4.3}$$

$$\begin{aligned}&\mathsf{terminatedAt}(punctuality(Id,\ nonPunctual),\ T) \leftarrow \\ &\qquad \mathsf{happensAt}(stopEnter(Id,\ StopId,\ scheduled),\ T).\end{aligned} \tag{4.4}$$

$$\begin{aligned}&\mathsf{initiatedAt}(drivingQuality(Id,\ low),\ T) \leftarrow \\ &\qquad \mathsf{initiatedAt}(punctuality(Id,\ nonPunctual),\ T), \\ &\qquad \mathsf{holdsAt}(drivingStyle(Id,\ unsafe),\ T).\end{aligned} \tag{4.5}$$

$$\begin{aligned}&\mathsf{initiatedAt}(drivingQuality(Id,\ low),\ T) \leftarrow \\ &\qquad \mathsf{initiatedAt}(drivingStyle(Id,\ unsafe),\ T), \\ &\qquad \mathsf{holdsAt}(punctuality(Id,\ nonPunctual),\ T).\end{aligned} \tag{4.6}$$

$$\begin{aligned}&\mathsf{terminatedAt}(drivingQuality(Id,\ low),\ T) \leftarrow \\ &\qquad \mathsf{terminatedAt}(punctuality(Id,\ nonPunctual),\ T).\end{aligned} \tag{4.7}$$

$$\begin{aligned}&\mathsf{terminatedAt}(drivingQuality(Id,\ low),\ T) \leftarrow \\ &\qquad \mathsf{terminatedAt}(drivingStyle(Id,\ unsafe),\ T).\end{aligned} \tag{4.8}$$

Clauses (4.1) and (4.2) state that a period of time for which vehicle Id is said to be *non-punctual* is initiated if it enters a stop later, or leaves a stop earlier than the scheduled time. Clauses (4.3) and (4.4) state that the period for which vehicle Id is said to be non-punctual is terminated when the vehicle arrives at a stop earlier than, or at the scheduled

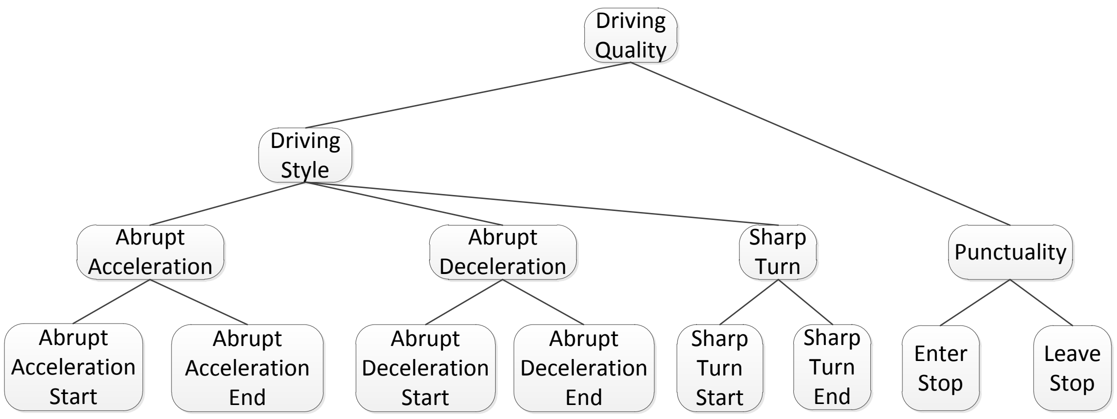

FIGURE 4.2: City Transport Management partial event hierarchy (we omit the whole hierarchy to save space). Additional high-level events, not presented here are *noise level*, *vehicle temperature*, and *passenger density*, which depend on corresponding low-level events and affect *driving quality*.

time. The definition of non-punctual vehicle uses two low-level events, $stopEnter$ and $stopLeave$.

Clauses (4.5)-(4.8) define *low driving quality*. Essentially, driving quality is said to be low when the driving style is unsafe and the vehicle is non-punctual. Driving quality is defined in terms of high-level events (we omit the definition of driving style to save space). Therefore, the bodies of the clauses defining driving quality include initiatedAt/2 and terminatedAt/2 literals.

4.2.1 ILED vs XHAIL

In this experiment, we tried to learn simultaneously definitions for all target concepts, a total of nine interrelated high-level events, seven of which are level-1, one is level-2 and one is level-3. The total number of low-level events is eleven, while for both high-level and low-level events, their negations are considered during learning. According to the employed language bias, the definitions of all high-level events must be learnt from data, while at the same time the high-level events may appear in the body of other high-level event definitions, in the form of (potentially negated) holdsAt/2, initiatedAt/2, or terminatedAt/2 predicates.

We used ten-fold cross validation with replacement, on small amounts of data, due to the complexity of the learning task. In each run of the cross validation, we randomly sampled 20 examples from the CTM dataset, 90% of which was used for training and 10% was retained for testing. This example size was selected after experimentation, in order for XHAIL to be able to handle it in an acceptable time frame. Each sample consisted of approximately 150 atoms (narrative and annotation). The examples were given to ILED in windows of granularity 5 and 10, and to XHAIL in one batch. Table 4.2

	ILED		XHAIL
	$G = 5$	$G = 10$	$G = 20$
Training Time (**hours**)	1.35	1.88	4.35
Hypothesis size	28.32	24.13	24.02
Revisions	14.78	13.42	–
Precision	63.344	64.644	66.245
Recall	59.832	61.423	62.567

TABLE 4.2: Comparative performance of ILED and XHAIL on selected subsets of the CTM dataset each containing 20 examples. G is the granularity of the windows.

presents the average training times, hypothesis size, number of revisions, precision and recall.

ILED took on average 1-2 hours to complete the learning task, for windows of 5 and 10 examples, while XHAIL required more than 4 hours on average to learn hypotheses from batches of 20 examples. Compared to activity recognition, the learning setting requires larger Kernel Set structures that are hard to reason with. An average Kernel Set generated from a batch of just 20 examples consisted of approximately 30-35 clauses, with 60-70 literals each.

Like the activity recognition experiments, precision and recall scores for ILED are comparable to those of XHAIL, with the latter being slightly better. Unlike the activity recognition experiments, precision and recall had a large diversity between different runs. Due to the complexity of the CTM dataset, the constructed hypotheses had a large diversity, depending on the random samples that were used for training. For example, some high-level event definitions were unnecessarily lengthy and difficult to be understood by a human expert. On the other hand, some level-1 definitions could, in some runs of the experiment, be learnt correctly even from a limited amount of data. Such definitions are fairly simple, consisting of one initiation and one termination rule, with one body literal in each case.

This experiment demonstrates several limitations of learning in large and complex applications. The complexity of the domain increases the intensity of the learning task, which in turn makes training times forbidding, even for a small amount of data such as 20 examples (approximately 150 atoms). This forces one to process small sets of examples at a time, which in complex domains like CTM, results in over-fitted theories and rapid increase in hypothesis size.

4.2.2 Learning With Hierarchical Bias

In an effort to improve the experimental results, we utilized domain knowledge about the event hierarchy in CTM and attempted to learn high-level events in different levels separately. To do so, we had to learn a complete definition for a level-n event from the entire dataset, before utilizing it as background knowledge in the learning process of a

	ILED		
level-1	$G = 10$	$G = 50$	$G = 100$
Training Time (min)	4.46 – 4.88	5.78 – 6.44	6.24 – 6.88
Revisions	2 – 11	2 – 9	2 – 9
Hypothesis size	4 – 18	4 – 16	4 – 16
Precision	100%	100%	100%
Recall	100%	100%	100%
level-2	$G = 10$	$G = 50$	$G = 100$
Training Time (min)	8.76	9.14	9.86
Revisions	24	17	17
Hypothesis size	31	27	27
Precision	100%	100%	100%
Recall	100%	100%	100%
level-3	$G = 10$	$G = 50$	$G = 100$
Training Time (min)	5.78	6.14	6.78
Revisions	6	5	5
Hypothesis size	13	10	10
Precision	100%	100%	100%
Recall	100%	100%	100%

TABLE 4.3: ILED with hierarchical bias.

level-$n+1$ event. To simplify the learning task further, we also used expert knowledge about the relation between specific low-level and high-level events, in order to exclude from the language bias mode declarations which were irrelevant to the high-level event that was being learnt at each time.

The experimental setting was therefore as follows: Starting from the level-1 target events, we processed the whole CTM dataset in windows of 10, 50 and 100 examples with ILED. Each high-level event was learnt independently of the others. Once complete definitions for all level-1 high-level events were constructed, they were added to the background knowledge. Then we proceeded with learning the definition of the single level-2 event (see Figure 4.2). Finally, after successfully constructing the level-2 definition, we performed learning in the top-level of the hierarchy, using the previously constructed level-1 and level-2 event definitions as background knowledge. We did not attempt a comparison with XHAIL because it could not handle the entire dataset.

Table 4.3 presents the results. For level-1 events, scores are presented as minimum-maximum pairs. For instance, the training times for level-1 events with windows of 10 examples, range from 4.46 to 4.88 minutes. Levels 2 and 3 have just one definition each, therefore Table 4.3 presents the respective scores from each run. Training times, hypotheses sizes and overall numbers of revisions are comparable for all levels of the event hierarchy. Level-1 event definitions were the easiest to acquire, with training times ranging approximately between 4.50 to 7 minutes. This was expected since clauses in level-1 definitions are significantly simpler than level-2 and level-3 ones. The level-2 event definition was the hardest to construct with training times ranging between 8 and 10 minutes, while a significant number of revisions was required for all window granularities. The definition of this high-level event (*drivingStyle*) is more complex than the simpler level-3 definition, for which training times are comparable to the ones of level-1 events.

Interestingly, we observed that most of the training time was devoted to checking an already correct definition against the part of the dataset that had not been processed yet. That is, for all target events, ILED converged to a complete definition in about 1.5 to 3 minutes after the initiation of the learning process. From that point on, the extra time was spent on testing the hypothesis against the new incoming data.

Window granularity affected somewhat the produced hypothesis for all target high-level events. Indeed, the definitions constructed with windows of 10 examples were slightly larger than the ones constructed with larger window sizes of 50 and 100 examples. Notably, the definitions constructed with windows of granularity 50 and 100, were found concise, meaningful and very close to the actual hand-crafted rules that were utilized in PRONTO.

4.3 Summary

In this chapter we presented an experimental evaluation for ILED, using real and synthetic data from two challenging real-life applications: Human activity recognition and transport management. The obtained results indicate that ILED scales adequately to large data volumes of sequential data with a time-like structure that is typical of event-based applications. It is significantly more efficient than XHAIL, without compromising the quality of the generated hypothesis in terms of predictive accuracy and hypothesis size. Moreover, we explored ILED's performance in more challenging event recognition applications (like the city transport management use case), that involve event hierarchies, and pointed-out some of ILED's limitations in such domains. Despite these limitations, our results indicate that by exploiting hierarchical bias, i.e. knowledge about the hierarchy, provided by domain experts, ILED is able to efficiently learn useful hypotheses even in these harder domains. In contrast, XHAIL could not handle the data volume in the city transport management experiment.

5 OLED: Online Learning of Event Definitions

In the previous chapter we presented a scalable, incremental strategy for learning event definitions in the form of Event Calculus theories from data that arrive over time. This strategy is designed to induce sound hypotheses and it minimizes the number of passes over the training set, that are required in order to learn a hypothesis. However, this learning strategy is inappropriate for many applications. First, real-life data comprise noise, therefore most of the times it is desirable to learn a less-than-perfect model with a good fit in the data, rather than learning nothing at all, since learning sound hypotheses from noisy data is not possible. Second, event recognition applications typically deal with continuous data flows, i.e. data that arrive at a high velocity, in potentially infinite streams. Methods that extract insights from such streams need to operate within tight memory and time constraints, building a decision model by a single pass over the training data [Gama, 2010; Gama and Gaber, 2007].

In this chapter we present OLED (Online Learning of Event Definitions), an ILP system that learns Event Calculus theories in a single pass over a data stream. OLED uses the Hoeffding bound [Hoeffding, 1963], a statistical tool that allows to build decision models using only a small subset of the data, by relating the size of this subset to a user-defined confidence level on the error margin of not making a (globally) optimal decision at a certain point during model construction [Dhurandhar and Dobra, 2012; Domingos and Hulten, 2000; Gama et al., 2011]. OLED learns a clause in a top-down fashion, by gradually adding literals to its body. Instead of evaluating each candidate specialization on the entire input, it accumulates training data from the stream, until the Hoeffding bound allows to select the best specialization. The instances used to make this decision are not stored or reprocessed, but discarded as soon as OLED extracts from them the necessary statistics for clause evaluation.

In the remainder of this chapter we discuss OLED in detail, starting with some necessary background and related work on learning from streams.

5.1 Learning from Data Streams

In this section we present some basic background on online learning for the purposes of this thesis. We begin with the differences between offline/online learning settings and the need for approximation algorithms for the latter. We then present a general-purpose strategy for online learning in propositional domains, which OLED "upgrades" to the relational case, based on approximating an "ideal", offline model by an online counterpart.

5.1.1 Learning From Data Streams

There are some fundamental differences between learning from static data and learning from data streams [Aggarwal, 2007, 2015; Chaudhry et al., 2006; Gaber et al., 2014; Gama, 2010; Leskovec et al., 2014; Muthukrishnan, 2005; Saitta, 2010]. While a learner may in principle spend unlimited time and memory processing data from a static dataset, it must operate within fixed time and memory resources when learning from a stream. And while in the offline scenario it can see the training examples as many times as needed, while learning online it must construct a model with a single pass over the data, as the data cannot be stored.

The limitations in the available resources make algorithms that extract insights from streams to resort to a trade-off between predictive accuracy and efficiency. Typically, online learners relax the requirement for exact/optimal models and isntead opt for good approximations of such models, utilizing techniques that allow for fast incorporation of new knowledge, using only a limited amount of resources [Gaber et al., 2014; Gama, 2010]. Such techniques typically combine *approximation* [Vazirani, 2013] with *randomization* [Motwani and Raghavan, 2010]. Approximate algorithms do not guarantee the best solution to a problem, but instead allow one to get as close as possible to such a solution (i.e. within an error margin ϵ) in a reasonable amount of time. In turn, randomized algorithms allow a probability δ of failure (not finding a solution). Techniques that combine approximation with randomization, often called (ϵ, δ)-approximations, provide solutions with probability $1 - \delta$, within an error margin of ϵ. Some examples of using (ϵ, δ)-approximation frameworks to learn from streams include frequent itemset mining with association rules [Manku and Motwani, 2002] and k-means clustering in data streams [Cormode et al., 2007].

One of the most successful (ϵ, δ)-approximation algorithms for learning from data streams is the HoeffdingTree algorithm used in the VDFT system for online learning of decision trees [Domingos and Hulten, 2000; Hulten et al., 2005]. The HoeffdingTree algorithm relies on the *Hoeffding bound* [Hoeffding, 1963] (Definition 5.1).

Definition 5.1 (The Hoeffding Bound). Let X be a random variable with range in $[0,1]$ and an observed mean $\overline{X}$ of its values after n independent observations. Then with probability $1-\delta$, the true mean $\hat{X}$ of the X variable lies in an interval $(\overline{X}-\epsilon, \overline{X}+\epsilon)$, where

$$\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$$

By associating the random variable X in Definition 5.1 with the outcome of an evaluation function that assesses a learner's performance in a machine learning context, the Hoeffding bound correlates the learner's loss ϵ with the size n of the training set. The larger the training set size, the smaller the error. Therefore, the Hoeffding bound provides a framework for devising scalable search heuristics, since it allows to map, with probability $1-\delta$, an arbitralily large input space to a small fraction of size $n = \mathcal{O}(\frac{1}{\epsilon^2}ln\frac{1}{\delta})$ [Gama, 2010], given an acceptable error margin ϵ.

Using a Hoeffding bound-based heuristic, a model may be constructed by using a limited (n-many) number of examples at each step of the training process, to make a decision that is an (ϵ,δ)-approximation of the optimal decision for that particular step. For instance, the HoeffdingTree algorithm learns decision trees in an online fashion, by using a Hoeffding bound-based heuristic to decide when (i.e. after how many examples) it should split a node on a particular attribute, thus generating new leaves in the tree. Moreover, thanks to the Hoeffding bound, the algorithm is able to guarantee that the learnt model is good approximation of its optimal counterpart, i.e. the online decision tree is not "too far away" (it does not significantly differ) from one that would be induced from infinite training examples. In addition to its usage for online learning of decision trees [Domingos and Hulten, 2000; Hulten et al., 2005, 2001], the Hoeffding bound has also been used for online clustering [Domingos and Hulten, 2001; Rodrigues et al., 2008], online learning of regression trees [Ikonomovska et al., 2011] and online decision rule learning [Gama et al., 2011; Kosina and Gama, 2012].

An additional attractive property of the Hoeffding bound is that it is *distribution-free* [Dhurandhar and Dobra, 2012], i.e. its validity is independent of the underlying probability distribution of the observations. This makes it a powerful tool for devising any-purpose, scalable search heuristics, although it comes with the price of the Hoeffding bound being more conservative than distribution-dependent bounds (i.e. it needs a larger number n of observations to reach the same (ϵ,δ)-approximation).

Numerous other distribution-free bounds have been proposed in machine learning, see [Langford, 2005] for a review. Such bounds have been used as estimators of classifiers' performance, providing (ϵ,δ)-approximations of the *generalization error*, i.e. the true, expected error of a classifier on the entire (possibly infinite) input, given its *empirical error*, i.e. the observable classifier's error on a data sample. In addition to the Hoeffding bound, prominent examples of distribution-free bounds include the Vapnik-Chervonenkis bounds (VC bounds) [Vapnik and Vapnik, 1998], Probably Approximately Correct Bayes bounds (PAC Bayes bounds) [McAllester, 1999], Occam Razor bounds [Blumer et al., 1990], Sample Compression bounds [Floyd and Warmuth, 1995] and Rademacher Complexity bounds [Bennett, 1962]. All these bounds have been studied theoretically and experimentally [Langford, 2005] and the results indicate that the Hoeffding bound (and the closely related Chernoff bound [Chernoff, 1952]) is much tighter than the aforementioned rival bound and provides a superior tool in approximating the true error given the empirical one in machine learning [Japkowicz and Shah, 2011; Langford, 2005; Shalev-Shwartz and Ben-David, 2014; Vovk et al., 2005].

5.2 Online Inductive Logic Programming

In this section we propose a general-purpose, online ILP approach, that uses a Hoeffding bound-based search heuristic to learn individual clauses in an online fashion. The goal, the details of which are presented in Section 5.3, is to properly adjust this general ILP framework to address the particular difficulties of learning Event Calculus programs, as discussed in the previous chapters of this thesis, towards an online ILP learner for event definitions.

As discussed in Section 2.3.2, ILP learners typically employ a separate-and-conquer strategy: clauses that cover subsets of the examples are constructed one by one recursively, until all examples are covered. Each clause is constructed in a top-down fashion, starting from an overly general clause and gradually specializing it by adding literals to its body. The process is guided by a heuristic function G that assesses the quality of each specialization on the entire training set. At each step, the literal (or set of literals) with the optimal G-score is selected and the process continues until certain criteria are met.

To adapt this core strategy to an online setting, we use the Hoeffding bound to evaluate candidate specializations on a subset of the training interpretations, instead of evaluating them on the entire input. To do so, we use an argument adapted from [Domingos and Hulten, 2000]. Let r be a clause and G a clause evaluation function with range in $[0, 1]$. The evaluation function that we use in this work will be discussed shortly. Assume also that after n training instances, r_1 is r's specialization with the highest observed mean G-score $\overline{G}$ and r_2 is the second best one, i.e. $\Delta\overline{G} = \overline{G}(r_1) - \overline{G}(r_2) > 0$. Then by the Hoeffding bound we have that for the true mean of the scores' difference $\Delta\hat{G}$ it holds $\Delta\hat{G} > \Delta\overline{G} - \epsilon$, with probability $1 - \delta$, where $\epsilon = \sqrt{\frac{ln(1/\delta)}{2n}}$. Hence, if $\Delta\overline{G} > \epsilon$ then $\Delta\hat{G} > 0$, implying that r_1 is indeed the best specialization to select at this point, with probability $1 - \delta$. In order to decide which specialization to select, it thus suffices to accumulate observations from the input stream until $\Delta\overline{G} > \epsilon$. Since ϵ decreases with the number of observations, given a desired δ, the number of observations n needed to reach a decision may be traded for a tolerable generalization error ϵ of not selecting the optimal specialization at a certain choice point. The observations need not be stored or reprocessed. We process each observation once to extract the necessary statistics for the computation of the G-score of each candidate specialization. This gives rise to a single-pass clause construction strategy.

Recall from Section 2.3.1 that throughout this thesis the *Learning from Interpretations* ILP setting is assumed, where each interpretation is independent form others [Blockeel et al., 1999]. This guarantees the independence of observations that is necessary for using the Hoeffding bound.

5.3 Online Learning of Event Calculus Theories

In this section we adapt the generic online ILP framework described in Section 5.2 towards online learning of event definitions in the form of Event Calculus programs.

We begin by relaxing the requirement for a hypothesis H to cover every training interpretation, in order to account for noise, and thus seek a theory with a good fit in the training data. To this end, we define true positive, false positive and false negative atoms as follows:

Definition 5.2 (TP, FP, FN atoms). Let B consist of the domain-independent EC axioms, r be a clause and I an interpretation. We denote by $narrative(I)$ and $annotation(I)$ the narrative and the annotation part of I respectively (see also Table 2.2(a)). We denote by M_I^r an answer set of $B \cup narrative(I) \cup r$. Given an annotation atom α we say that:

- α is a true positive (*TP*) atom w.r.t. clause r iff $\alpha \in annotation(I) \cap M_I^r$.
- α is a false positive (*FP*) atom w.r.t. clause r iff $\alpha \in M_I^r$ but $\alpha \notin annotation(I)$.
- α is a false negative (*FN*) atom w.r.t. clause r , iff $\alpha \in annotation(I)$ but $\alpha \notin M_I^r$.

□

5.3.1 Evaluating Clauses

We seek a theory H that maximizes the *TP* atoms, while minimizing the *FP* and *FN* atoms, collectively for all its clauses. To do so, we maintain a count per clause for each such atom. For an initiatedAt clause, its *TP* (resp. *FP*) count is increased each time it correctly (resp. incorrectly) initiates a complex event (according to the annotation). For a terminatedAt clause, its *TP* count is increased each time it correctly allows a complex event to persist, by not terminating it. Its *FN* count is increased when it incorrectly terminates a complex event.

As discussed in previous chapters, when learning structure in Horn logic with ILP, a theory H is augmented with new clauses to increase its total *TP* count, while existing clauses in H are specialized to decrease the *FP* count. This strategy is in principle not applicable to the problem at hand, as demonstrated by Example 2.4 of Section 2.4.2. When learning domain-specific axioms in the Event Calculus, the addition of new clauses may be necessary to eliminate *FPs*, while clause specialization may be necessary to increase *TPs*, as detailed below. Given a theory H and an interpretation I, assume that $B \cup H$ does not cover I. Then one of the following holds:

1. **The *FN* case.** There is at least one *FN* atom α. This may be due to one of the following:
   (a) No initiatedAt clause in H "fires", failing to initiate the complex event that corresponds to α, when it should. In this case, generating a new initiatedAt clause, eliminates the *FN* atom, turning it into a *TP*.
   (b) One or more terminatedAt clauses in H are over-general, terminating the complex event that corresponds to α when they should not. Specializing the over-general clauses, eliminates the *FN* atom, turning it into a *TP*.
2. **The *FP* case.** There is at least one *FP* atom α. This may be due to one of the following:
   (a) No terminatedAt clause in H "fires", failing to terminate the complex event that corresponds to α when it should, so α erroneously persists by inertia. Generating a new terminatedAt clause eliminates the *FP*.
   (b) One or more initiatedAt clauses are over-general, re-initiating a corresponding complex event when they should not. Specializing the over-general clauses eliminates the *FP*.

Combining the above observations we derive an overall strategy for heuristically driving the generation of a good hypothesis H. We try to maximize both precision and recall for H as a whole. To improve precision, we either specialize existing initiatedAt clauses, which are already in H, or we generate (add to H) new terminatedAt clauses, since both these actions reduce the total *FP* count for H. Dually, to improve recall we either specialize existing terminatedAt clauses, or we generates new initiatedAt clauses, since both these actions reduce the total *FN* count, while increasing the total *TP* count. Since in this strategy existing clauses (clauses already in H) are specialized to improve precision and recall in the case of initiatedAt and terminatedAt clauses respectively, we derive the following scoring function for such clauses:

Definition 5.3 (Clause evaluation function)**.** Let us denote by TP_r, FP_r and FN_r respectively, the accumulated *TP, FP* and *FN* counts of clause r over the input stream. The

Algorithm 5 OnlineLearning$(\mathcal{I}, B, G, \delta, d, N_{min}, S_{min})$

Input: $\mathcal{I}$: A stream of training interpretations; B: Background knowledge; G: Clause evaluation function; δ : Confidence for the Hoeffding test; d : Specialization depth; S_{min} : Clause G-score quality threshold.

```
 1: $H := \emptyset$
 2: **for all** $I \in \mathcal{I}$ **do**
 3:     Update $TP_r, FP_r, FN_r$ and $N_r$ counts from $I$, for each $r \in H$ and each $r' \in \rho_d(r)$,
        where $N_r$ denotes the number of examples on which $r$ has been evaluated so far.
 4:     **if** ExpandTheory$(B, H, I)$ **then**
 5:         $H \leftarrow H \cup$ StartNewClause$(B, I)$
 6:     **else**
 7:         **for all** clause $r \in H$ **do**
 8:             $r \leftarrow$ ExpandClause$(r, G, \delta)$
 9:     $H \leftarrow$ Prune$(H, S_{min})$
10: **return** $H$
11: **function** StartNewClause$(B, I)$:
12:     Generate a bottom clause $\perp$ from $I$ and $B$
13:     $r := head(\perp) \leftarrow$
14:     $\perp_r := \perp$
15:     $N_r = FP_r = TP_r = FN_r := 0$
16:     **return** $r$
17: **function** ExpandClause$(r, G, \delta)$:
18:     Compute $\epsilon = \sqrt{\frac{ln(1/\delta)}{2N_r}}$ and let $\overline{G}$ denote the mean value of a clause's $G$-score
19:     Let $r_1$ be the best specialization of $r$, $r_2$ the second best and $\Delta\overline{G} = \overline{G}(r_1) - \overline{G}(r_2)$
20:     Let $\tau$ equal the mean value of $\epsilon$ observed so far
21:     **if** $\overline{G}(r_1) > \overline{G}(r)$ **and** $[\Delta\overline{G} > \epsilon$ **or** $\tau < \epsilon]$:
22:         $\perp_{r_1} := \perp_r$
23:         **return** $r_1$
24:     **else return** $r$
25: **function** prune$(H, S_{min})$:
26:     **let** $k$ be the average number of examples $N_r$, for all $r \in H$, that have been used so far by
        ExpandClause, in order to expand clauses to their best-scoring specialization
27:     **for all** $r \in H$ **do**
28:         **if** $r$ has not "changed" for $k' \geq k$ examples **then**
29:             **if** $S_{min} - \overline{G}(r) > \epsilon$, where $\epsilon$ is the current Hoeffding bound **then**
30:                 $H \leftarrow H \setminus r$
31:     **return** $H$
```

clause evaluation function G for a clause r is a function with range in $[0, 1]$ defined as follows:

$$G(r) = \begin{cases} \overbrace{\dfrac{TP_r}{TP_r + FP_r}}^{\text{precision}} & \text{if } r \text{ is an initiatedAt clause} \\[2ex] \underbrace{\dfrac{TP_r}{TP_r + FN_r}}_{\text{recall}} & \text{if } r \text{ is a terminatedAt clause} \end{cases} \qquad \square$$

Process	Cause of Failure	Action	Justification
L_{init}	*FP*	Clause expansion	Case 2(b)
L_{init}	*FN*	Theory expansion	Case 1(a)
L_{term}	*FP*	Theory expansion	Case 2(a)
L_{term}	*FN*	Clause expansion	Case 1(b)

TABLE 5.1: Action dispatching scheme for OLED's initiatedAt (L_{init}) and terminatedAt (L_{term}) parallel processes.

5.3.2 The OLED system

In this section we discuss the main functionality of OLED, presented in Algorithm 5, in detail. Learning begins with an empty hypothesis H. On the arrival of new interpretations, OLED either expands H, by generating a new clause, or tries to expand (specialize) an existing clause. Clauses of low quality are pruned, after they have been evaluated on a sufficient number of examples. Each incoming interpretation is processed once, to extract the necessary statistics for clause evaluation in the form of *TP*, *FP* and *FN* counts, and is subsequently discarded.

To distinguish between the different cases presented in Section 5.3.1, initiation and termination clauses are learnt separately in parallel, by two processes L_{init} and L_{term} respectively (each of these processes runs separately Algorithm 5). The input stream is forwarded to both of these processes at the same time. Thanks to this decoupling, when either process fails to account for a training interpretation, it is able to infer the causes of failure in terms of *FP* and *FN* atoms. In particular L_{init} detects *FP*/*FN*-failures based on cases 2(b)/1(a) respectively and L_{term} detects *FP*/*FN*-failures based on cases 2(a)/1(b). Depending on the cause of failure, the process dispatches control either to the theory expansion, or the clause expansion subroutines. The choice among these actions is made by the boolean function `ExpandTheory` in line 4 of Algorithm 5. Action selection is based on the analysis of Section 5.3.1 and summarised in Table 5.1. Below we present an example for illustration purposes.

Example 5.1. *Initially, processes L_{init} and L_{term} start with two empty hypotheses, H_{init} and H_{term}. Assume that the annotation in one of the incoming interpretations dictates that the* moving *complex event holds at time* 10*, while it does not hold at time* 9*. Since no clause in H_{init} yet exists to initiate* moving *at time* 9*, L_{init} detects the* moving *instance at time* 10 *as an* FN *and proceeds to theory expansion (second case in Table 5.1), generating an initiation clause for* moving*. L_{term} is not concerned with initiation conditions, so it will take no actions in this case. Then, a new interpretation arrives, where the annotation dictates that* moving *holds at time* 20*, but does not hold at time* 21*. In this case, since no clause yet exists in H_{term} to terminate* moving *at time* 20*, L_{term} will detect an* FP *instance at time* 21*. It will then proceed to theory expansion (third case in Table 5.1), generating a new termination condition for moving. At the same time, assume that the initiation clause in H_{init} is over-general and erroneously re-initiates* moving *at time* 20*, generating an FP instance for the*

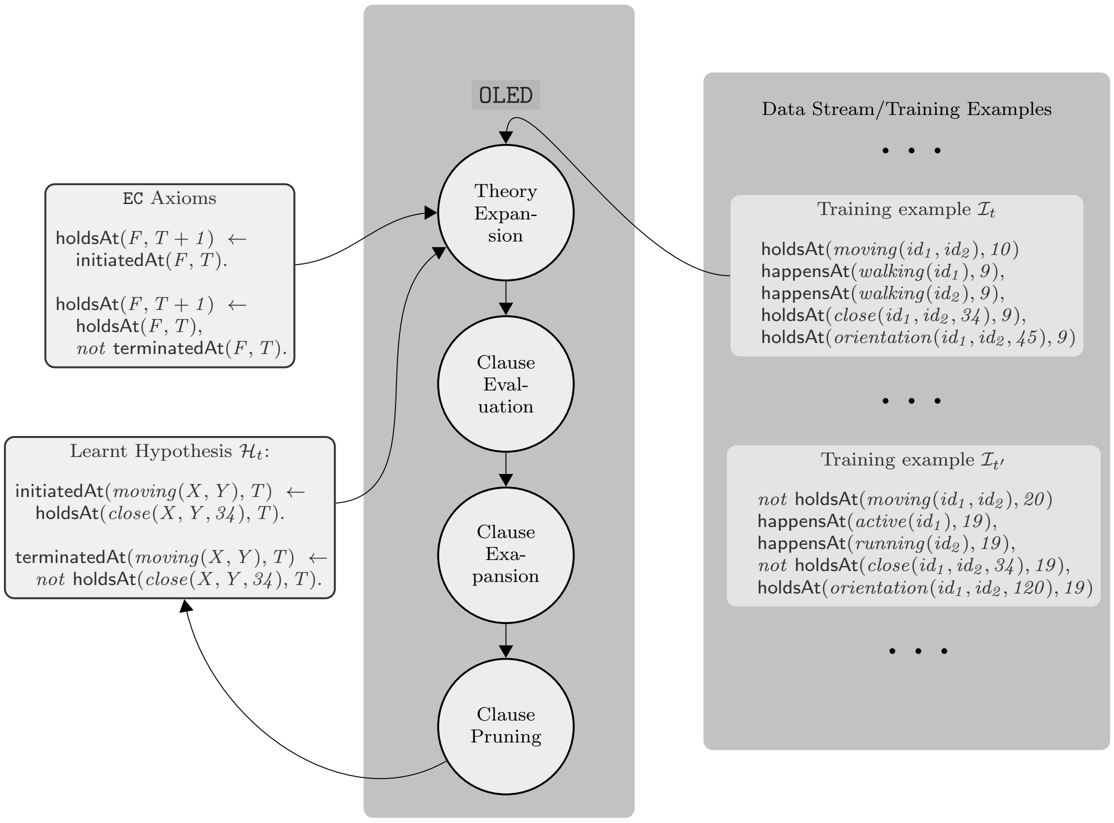
OLED
Theory Expansion
Clause Evaluation
Clause Expansion
Clause Pruning
EC Axioms
holdsAt(F, T + 1) ← initiatedAt(F, T).
holdsAt(F, T + 1) ← holdsAt(F, T), not terminatedAt(F, T).
Learnt Hypothesis $\mathcal{H}_t$:
initiatedAt(moving(X, Y), T) ← holdsAt(close(X, Y, 34), T).
terminatedAt(moving(X, Y), T) ← not holdsAt(close(X, Y, 34), T).
Data Stream/Training Examples
Training example $\mathcal{I}_t$
holdsAt(moving(id1, id2), 10) happensAt(walking(id1), 9), happensAt(walking(id2), 9), holdsAt(close(id1, id2, 34), 9), holdsAt(orientation(id1, id2, 45), 9)
Training example $\mathcal{I}_{t'}$
not holdsAt(moving(id1, id2), 20) happensAt(active(id1), 19), happensAt(running(id2), 19), not holdsAt(close(id1, id2, 34), 19), holdsAt(orientation(id1, id2, 120), 19)

FIGURE 5.1: Illustration of OLED's learning process.

L_{init} process at time 21. In response to that, L_{init} will proceed to clause expansion (first case in Table 5.1), penalizing the over-general initiation clause by increasing its FP *count, thus contributing towards its potential replacement by one of its specializations.*

In the remainder of this section, we go into the details of theory and clause expansion, as well as other interesting aspects of OLED's functionality, which is illustrated in Figure 5.1.

Theory Expansion. The theory expansion process is handled by the `StartNewClause` function in Algorithm 5. A new clause is generated in a data-driven fashion, by constructing a bottom clause $\perp$ [Muggleton, 1995b] from a training interpretation. Theory expansion consists of the addition of the empty-bodied clause $r = head(\perp) \leftarrow$ to theory H. From that point on, r is gradually specialized by the addition of literals from $\perp$ to its body. We denote by $\perp_r$ the bottom clause associated to clause r. Figure 5.2 illustrates the generation of a new clause from a bottom clause.

As discussed in Section 2.3.2, in a typical ILP setting, a bottom clause is constructed by selecting a target predicate instance e as a "seed", placing it in the head of a newly generated clause $\perp$ with an empty body. A set of atoms that follow deductively from e and the background knowledge are placed in the body of $\perp$. Constants in $\perp$ are replaced by variables, where appropriate, as indicated by a particular language bias, which is

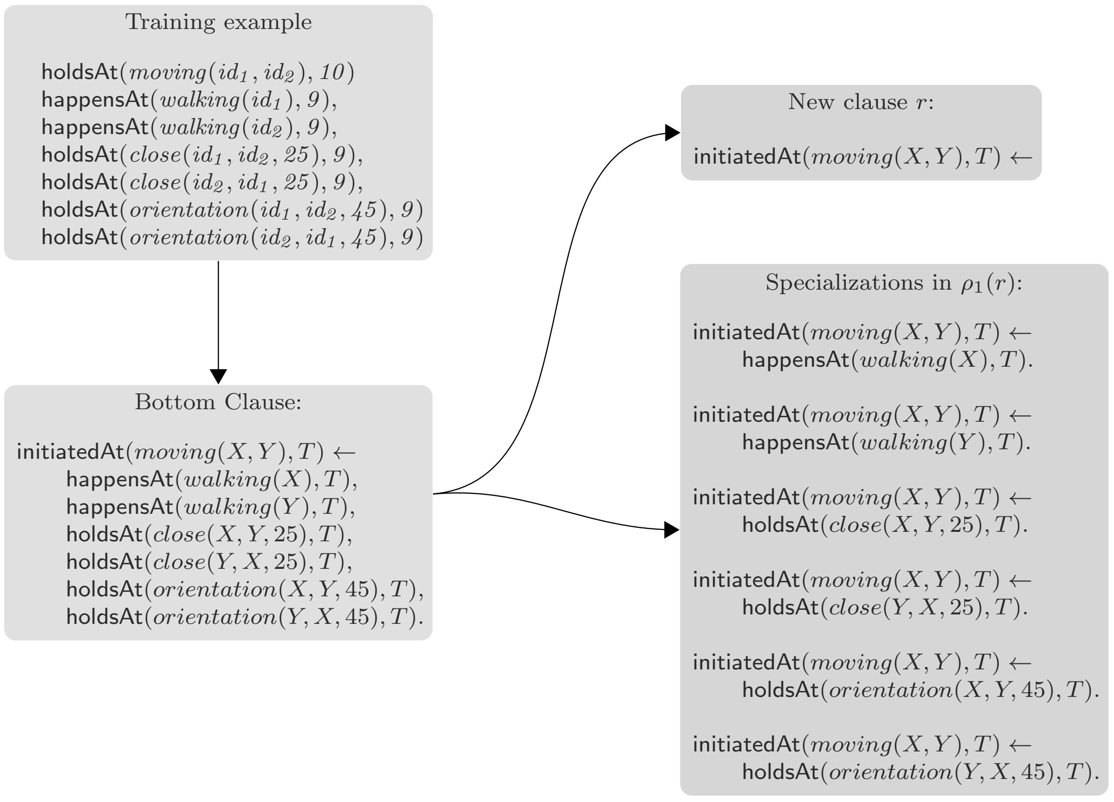
Training example
holdsAt($moving(id_1, id_2), 10$)
happensAt($walking(id_1), 9$),
happensAt($walking(id_2), 9$),
holdsAt($close(id_1, id_2, 25), 9$),
holdsAt($close(id_2, id_1, 25), 9$),
holdsAt($orientation(id_1, id_2, 45), 9$)
holdsAt($orientation(id_2, id_1, 45), 9$)
Bottom Clause:
initiatedAt($moving(X, Y), T$) $\leftarrow$
happensAt($walking(X), T$),
happensAt($walking(Y), T$),
holdsAt($close(X, Y, 25), T$),
holdsAt($close(Y, X, 25), T$),
holdsAt($orientation(X, Y, 45), T$),
holdsAt($orientation(Y, X, 45), T$).
New clause r:
initiatedAt($moving(X, Y), T$) $\leftarrow$
Specializations in $\rho_1(r)$:
initiatedAt($moving(X, Y), T$) $\leftarrow$ happensAt($walking(X), T$).
initiatedAt($moving(X, Y), T$) $\leftarrow$ happensAt($walking(Y), T$).
initiatedAt($moving(X, Y), T$) $\leftarrow$ holdsAt($close(X, Y, 25), T$).
initiatedAt($moving(X, Y), T$) $\leftarrow$ holdsAt($close(Y, X, 25), T$).
initiatedAt($moving(X, Y), T$) $\leftarrow$ holdsAt($orientation(X, Y, 45), T$).
initiatedAt($moving(X, Y), T$) $\leftarrow$ holdsAt($orientation(Y, X, 45), T$).

FIGURE 5.2: Generation of a new clause r and its specializations in $\rho_1(r)$.

typically mode declarations (see also Section 2.3.2). To find a clause with a good fit in the data, a refinement operator ρ is used to generate candidate clauses that θ-subsume $\perp$.

Due to the fact that learning domain-specific axioms in the Event Calculus falls in the non-Observational Predicate Learning class of problems (see Section 2.4.1), the aforementioned approach cannot be used directly. As in the case of XHAIL and ILED, OLED uses abduction as a workaround in order to obtain the missing target predicate instances and then construct bottom clauses from them. The process has been detailed in previous sections in this thesis, e.g. see Sections 2.4.1 and 2.4.3, as well as Example 2.2.

OLED is an any-time algorithm, i.e. it may output the hypothesis constructed so far at any time during the learning process. We allow a "warm-up" period, in the form of a minimum number of training instances N_{min} on which a clause r must be evaluated before it can be included in an output hypothesis.

Clause Expansion. We use the Hoeffding bound to select among competing specializations of a clause r. These specializations are generated by adding one or more literals from $\perp_r$ to the body of r. An input parameter d for *specialization depth* serves as an upper bound to the number of literals that may be added each time. We use $\rho_d(r)$ to denote the set of specializations for clause r. Formally:

$$\rho_d(r) = \begin{cases} \{head(r) \leftarrow\} & \text{if } d = 0 \\ \\ \{head(r) \leftarrow body(r) \wedge D \mid D \subset body(\bot_r) \text{ and } |D| \leq d\} & \text{else} \end{cases}$$

For instance, $\rho_1(r)$ consists of all "one-step" specializations of r (i.e. those that result by the addition of a single literal from $\bot_r$), while $\rho_2(r)$ consists of $\rho_1(r)$ plus all "two-step" specializations, and so on. Figure 5.2 illustrates the specializations in $\rho_1(r)$ for an empty-bodied clause r.

While specializing a clause r, it may be wasteful to evaluate specializations that result by the addition of certain literals to the body of r, since such specializations are of worst quality than the parent clause r. However, such literals may be necessary in the specialization process for the introduction of fresh variables in the clause, and while adding them alone to r does not yield any gain, adding them in conjunction with other literals may yield a specialization of high quality. As an example, consider the clause that results by adding the literal $before(T_2, T_1)$ to the empty-bodied clause $\mathsf{initiatedAt}(fluent, T_1) \leftarrow$. Although the clause

$$\begin{aligned} &\mathsf{initiatedAt}(fluent, T_1) \leftarrow \\ &\quad before(T_2, T_1). \end{aligned}$$

does not make much sense, using $before/2$ in the specialization process might be necessary for introducing the fresh variable T_2, and using $before(T_2, T_1)$ in conjunction with another literal, may yield a good clause, e.g.

$$\begin{aligned} &\mathsf{initiatedAt}(fluent, T_1) \leftarrow \\ &\quad before(T_2, T_1), \\ &\quad \mathsf{happensAt}(event, T_2). \end{aligned}$$

Therefore, it is helpful to be able to perform multiple specialization steps at once in order to obtain a clause of better quality than the current one. To allow for that, OLED supports "look-ahead specifications" [Blockeel and De Raedt, 1998], i.e. user-defined directives that instruct the system to try particular literals in conjunction, thus searching "deeper" in the specialization lattice. For this to happen, the corresponding literals need to be found together in the bottom clause that is used as a search space.

A clause r is expanded, i.e. replaced by its best-scoring specialization from $\rho_d(r)$, when a sufficient number of interpretations have been seen, for which $\Delta\overline{G} > \epsilon$, as described in Section 5.2, where $\Delta\overline{G}$ is the observed difference between the mean G-scores of r's best and second best specializations and ϵ is given from Definition 5.1. To ensure that no clause r is replaced by a specialization of lower quality, r itself is also considered as a potential candidate along with its specializations from $\rho_d(r)$. This ensures that expanding a clause to its best-scoring specialization is better, with probability $1-\delta$, than not expanding it at all.

An important difference between learning from static and streaming data is that, while in the former case a learner may have random access to the training data, in the latter case it is obliged to process the data sequentially, in the order in which they arrive [Gaber et al., 2014]. As a result, online learners are typically subject to order effects, i.e. they are sensitive to the order in which the examples are presented. Using the Hoeffding bound allows OLED to mitigate such effects, since clause expansion is postponed until sufficient evidence for the quality of the candidate specializations is provided by the data.

Clause evaluation. To allow for clause evaluation, OLED maintains statistics for each clause $r \in H$ and each of its specializations in $\rho_d(r)$, over the training sequence, as described above. The sufficient statistics are the cumulative TP_r, FP_r and FN_r counts per clause, in order to compute the G-score (Definition 5.3) for each clause r. Additionally, a count N_r of the training instances on which clause r has been evaluated so far is maintained, in order to calculate the Hoeffding bound (Definition 5.1). The TP_r, FP_r, FN_r and N_r counts are updated for each clause r, whenever a new training interpretation arrives.

Tie-breaking. When the scores of two or more specializations are very similar, a large number of training instances may be required to decide between them. This could be wasteful, since any one of the specializations may be chosen. In such cases, as in [Domingos and Hulten, 2000], we break ties as follows: Instead of waiting until $\Delta\overline{G} > \epsilon$, as required by the Hoeffding bound-based heuristic, we expand r to its best-scoring specialization if $\Delta\overline{G} < \epsilon < \tau$, where τ is a tie-breaking threshold. Recall that ϵ decreases with the number n of training examples, thus it may fall below τ. We follow [Yang and Fong, 2011] and use an adaptive tie-breaking threshold, set to the mean value of ϵ that has been observed so far in the training process (see line 20, Algorithm 5). In the case of a tie between r itself and its best-scoring specialization, we follow a conservative approach and do not expand r, i.e. such ties are broken in favor of the parent clause. To avoid breaking ties between two equally good specializations r_1 and r_2 too early, we allow a "grace period" in tie-breaking, in the form of a minimum number of examples N_{min} on which r_1 and r_2 must have been evaluated before we select one of them. The problem of breaking ties too early may occur for instance in the case where $\overline{G}(r_1) = \overline{G}(r_2)$, after evaluating r_1 and r_2 on only a very small number of examples. Then $\Delta\overline{G} = 0 < \tau$, so based on the heuristic mentioned above, there is a tie between r_1 and r_2. However, this is misleading because if r_1 and r_2 are evaluated on a larger number of examples, the situation between them may change.

Clause pruning. Often, bad clauses may be constructed, whose quality cannot be improved. This could happen when e.g. a clause has been learnt from a noisy example. Maintaining these clauses and constantly evaluating them on new examples is pointless and wasteful. To address this issue, OLED supports the removal of low-quality clauses during the learning process. This happens when a low-quality clause r does not "change" for a long period of time. This means either that r cannot be specialized any further (it has exhausted all antecedent literals from its bottom clause), or that none of its available specializations yields any significant gain, therefore the specialization process is "stuck". More formally, online clause pruning is implemented as follows: Let n be the number of examples that suffice to decide when to expand a clause to its best-scoring specialization, according to the Hoeffding bound-based heuristic, and let k be the average value of such n's, observed so far in the training process. If a clause r does not "change" (does not get specialized) after $k' \geq k$ examples, and its quality is low, then it is removed. To decide if a clause is of low quality, we also use the Hoeffding bound: Given a quality threshold S_{min}, at the point when $S_{min} - \overline{G}(r) > \epsilon$, where ϵ is given by Definition 5.1, we have that with probability $1 - \delta$, the true mean of r's G-score is lower than the quality threshold S_{min}. Therefore r should be removed.

5.4 Discussion and Related Work

The need for deriving Hoeffding-like bounds for relational data has been identified in the field of Statistical Relational Learning [Dhurandhar and Dobra, 2012; Getoor, 2007], since such bounds can be used as the basis for sampling algorithms. Given the above discussion, an additional application of such bounds is the development of scalable learning algorithms in relational domains. However, deriving such bounds for relational data is not straightforward. The reason is that distribution-free bounds require an *independently and identically distributed* (*idd*) assumption on the data, which is omnipresent in propositional domains, but does not hold in principle in structured data, due to relational dependencies between the variables. This issue has been addressed in the literature [Dhurandhar and Dobra, 2010, 2012; Jensen, 1999; Jensen and Neville, 2002], deriving necessary and sufficient conditions on the underlying structure in order to support the *iid* assumption for some relational domains. In the general case, however, using Hoeffding-like bounds to derive (ϵ, δ)-approximations in relational learning is under-explored.

An early approach to use a Hoeffding bound-based heuristic for scalable relational learning is presented in [Hulten et al., 2003], where the VFREL system is presented. VFREL scales to large data volumes by identifying the relations that are important to the learning task and focusing on these important relations, hence saving time by ignoring ones that are not important. This is achieved by incorporating a sampling mechanism based on the Hoeffding bound that identifies the important relational attributes with a single scan of the data. Therefore, the Hoeffding bound in [Hulten et al., 2003] is actually used for propositional feature selection and it does not actually deal with relational data.

An ILP approach that uses the Hoeffding bound for relational learning is HTILDE [Lopes and Zaverucha, 2009], an extension of the TILDE system for learning first-order decision trees [Blockeel and De Raedt, 1998]. These are decision trees where each internal node consists of a conjunction of literals and each leaf is a propositional predicate representing a class. TILDE constructs trees by testing conjunctions of literals at each node, using an ILP refinement operator to generate the conjunctions and information gain as the guiding heuristic. HTILDE extends TILDE by using the Hoeffding bound to perform these internal tests on a subset of the training data. To ensure independence of observations, HTILDE learns from interpretations [Blockeel et al., 1999], a setting, used also by OLED, where each training instance is assumed a disconnected part of the dataset.

Like TILDE, HTILDE learns clauses with a propositional predicate in the head (representing a class). However, the head of a complex event definition is typically a first-order predicate, containing variables that appear in the body of the clause and express relations between entities. Therefore, HTILDE is not general enough for the problem that we address in this work. Additionally, HTILDE requires a fully annotated dataset, while in the setting we assume here, annotation for target predicates is missing.

6 Experimental Evaluation for OLED

In this chapter we evaluate OLED on CAVIAR, the benchmark dataset for activity recognition and compare it to a number of batch learning techniques. We obtain results of comparable predicative accuracy with significant speed-ups in training time. We also show that OLED outperforms hand-crafted rules for the particular domain and matches the performance of ILED, which is a sound incremental learner that can only operate on noise-free datasets. All experiments were conducted on a Linux machine with a 3.6GHz processor (4 cores and 8 threads) and 16GiB of RAM. The code and data are available online[1].

6.1 Comparison with Manually Constructed Rules and Batch Learning

The purpose of this experiment was to assess whether OLED is able to efficiently learn theories of comparable quality to hand-crafted rules and state-of-the-art batch learning approaches. We compare OLED to the following: (i) EC_{crisp}, a hand-crafted set of clauses for the CAVIAR dataset, described in [Artikis et al., 2010c]; (ii) EC_{MM} [Skarlatidis et al., 2015], a probabilistic version of EC_{crisp} with weights learnt by the Max-Margin weight learning method for Markov Logic Networks (MLNs) of [Huynh and Mooney, 2009]; (iii) XHAIL [Ray, 2009b], a hybrid abductive-inductive learner capable of learning programs in the EC, discussed in detail in Section 2.4.3. EC_{MM} was selected because it was shown to achieve good results on CAVIAR [Skarlatidis et al., 2015]. XHAIL was selected as one of the few ILP systems that is able to learn theories in the EC. OLED and XHAIL were implemented using the Clingo[2] answer set solver as the core reasoning component, while the EC_{MM} approach used in this experiment was implemented in the LoMRF framework[3] for MLNs.

To evaluate EC_{MM}, [Skarlatidis et al., 2015] used a fragment of the CAVIAR dataset, which is also the one we use in this experiment. The target complex events in this

[1] `https://github.com/nkatzz/OLED`
[2] `http://potassco.sourceforge.net/`
[3] `https://github.com/anskarl/LoMRF`

		Method	Precision	Recall	F_1-score	Theory size	Time (sec)
(a)	*Move*	EC_{crisp}	**0.909**	0.634	0.751	28	–
		EC_{MM}	0.844	0.941	**0.890**	28	1692
		XHAIL	0.779	0.914	0.841	**14**	7836
		OLED	0.709	**0.948**	0.812	34	**12**
	Meet	EC_{crisp}	0.687	0.855	0.762	23	–
		EC_{MM}	0.919	0.813	**0.863**	23	1133
		XHAIL	0.804	**0.927**	0.861	**15**	7248
		OLED	**0.943**	0.750	0.836	29	**23**
(b)	*Move*	EC_{crisp}	**0.721**	0.639	0.677	28	–
		OLED	0.653	**0.834**	**0.732**	42	124
		EC_{crisp}	0.644	0.855	0.735	23	–
	Meet	OLED	**0.678**	**0.953**	**0.792**	30	107
(c)	*Move*	ILED	0.947	**0.981**	**0.963**	55	**34**
		OLED	**0.963**	0.934	0.948	**31**	35
	Meet	ILED	0.930	**0.976**	0.952	65	**30**
		OLED	**0.975**	0.933	**0.953**	**53**	42

TABLE 6.1: Experimental results for OLED from the CAVIAR dataset

dataset are related to two persons *meeting each other* or *moving together* and the training data consists of the parts of CAVIAR that involve these complex events. The fragment dataset contains a total of 25738 training interpretations. There are 6272 interpretations in which *moving* occurs and 3722 in which *meeting* occurs. OLED's results were achieved using significance $\delta = 10^{-5}$, a clause pruning threshold S_{min} of 0.7 for *meeting* and 0.5 for *moving* and specialization depth parameter $d = 2$ for *meeting* and $d = 1$ for *moving*. The results reported with these parameter configurations are the best among several other parameter settings that we tried for S_{min} and d. The reported training time for each run of OLED was the maximum training time of the two processes that learn initiation and termination clauses in parallel.

Results were obtained using 10-fold cross validation and are presented in Table 6.1(a) in the form of *precision, recall* and f_1*-score*. These statistics were micro-averaged over the instances of recognized complex events from each fold of the 10-fold cross validation process. That is, the *TP, FP* and *FN* counts from each fold were summed, and precision, recall and f_1-score were calculated using these sums. Table 6.1(a) also presents average training time per fold for all approaches except EC_{crisp} (where no training is involved), average theory sizes (total number of literals) for OLED and XHAIL, as well as the fixed theory size of EC_{crisp} and EC_{MM}. Results for EC_{MM} were obtained using MAP inference.

EC_{MM} achieves the best f_1-score for both complex events, followed closely by XHAIL. OLED achieves a comparable predictive accuracy (particularly for *meeting*), while it outscores the hand-crafted rules. Moreover, OLED achieves speed-ups of several orders of magnitude as compared to EC_{MM} and XHAIL, due to its single-pass strategy. The superior performance of EC_{MM} and XHAIL is due to them being batch learners, optimizing their respective outcomes over the entire training set. This also explains their increased training times. Regarding theory size, XHAIL learns significantly more compressed hypotheses than OLED. The reason is that XHAIL learns whole theories, while OLED learns each clause separately to gain in efficiency.

6.2 Activity Recognition on the Entire CAVIAR Dataset

We also present experimental results from running OLED on the entire CAVIAR dataset, which contains a total of 282067 training interpretations. The target complex events are *meeting* and *moving* as above. The number of positive interpretations for both complex events is also the same as above, since the data fragment used in the previous experiment contains the parts of CAVIAR where these complex events occur. In contrast, the number of negative training instances is much larger in this experiment.

Due to high training times for XHAIL and $\mathsf{EC}_{\mathsf{MM}}$, we do not present results with these approaches and we compare OLED only to the set of manually developed clauses $\mathsf{EC}_{\mathsf{crisp}}$. The experimental setting was as follows: We used 10-fold cross validation over the fragment used in the previous experiment, but in each fold, the respective training and testing sets were augmented by a number of negative training sequences. In particular, in each fold, 90% of the negative training sequences from the remaining part of CAVIAR (i.e. the part not contained in the data fragment of the previous experiment) was added to the training set of the fold, while the remaining 10% was added to the test set. The parameter configuration for OLED was the same as in the previous experiment, with the exception of the specialization depth for *meeting*, which was set to $d = 1$, since the value of $d = 2$ used for *meeting* in the previous experiment increased training time without any significant gain in the quality of the outcome.

Table 6.1(b) shows the results. As previously, statistics were micro-averaged over the recognized complex event instances from each fold. Both approaches' performance is decreased, as compared to the previous experiment, due to the increased number of false positives, caused by the large number of additional negative instances. OLED still outscores the hand-crafted knowledge base.

6.3 Comparison with an Incremental Learner

We also compared OLED to ILED, the incremental learner that is based on the methodology of the XHAIL system and is able to learn theories in the EC [Katzouris et al., 2015]. Recall form Chapter 3 that ILED works by revising past hypotheses to account for new examples that arrive over time. In contrast to OLED, a revised hypothesis must account for all past training instances. ILED has a scalable revision strategy that requires at most one pass over past examples to revise a hypothesis. However, this strategy is based on the assumption that the data is noise-free, and therefore ILED cannot be used with CAVIAR, which exhibits various types of noise – see [Artikis et al., 2010c] for details.

In order to compare the two systems we thus generated a noise-free version of CAVIAR with artificial annotation for the *moving* and *meeting* complex events. To produce the annotation, we used the hand-crafted knowledge base $\mathsf{EC}_{\mathsf{crisp}}$ for inference over the CAVIAR narrative. The dataset contains a total of 282067 training interpretations. From these, 6172 are positive interpretations for *meeting* and 5204 are positive interpretations for *moving*. We used 10-fold cross validation to assess the performance of the the compared systems. For each fold, the training (resp. test) set consisted of 90% (resp. 10%) of positive and negative interpretations for each complex event. The input parameter configuration for OLED was as reported in the experiment of Section 6.2.

The results are presented in Table 6.1(c). As previously, results were micro-averaged over each fold, while training times and hypotheses sizes are averages from the separate runs. The predictive accuracy for both systems, in terms of their respective f_1-scores, is comparable, with ILED's being slightly better. This was expected, since ILED re-scans the historical memory of past data to revise its theories. Training times are also comparable, with OLED's being slightly higher, as compared to ILED's. ILED is able to avoid certain computations by inferring that they are redundant, based on the assumption that the data is noise-free. Regarding theory size, OLED learns significantly shorter hypotheses that ILED. OLED prunes a number of its learnt clauses, in an effort to avoid fitting potential noise in the data and also follows a conservative clause expansion strategy. In contrast, ILED tries to account for every positive example in the training data (and exclude every negative one), since it is designed for learning sound hypotheses.

6.4 Scalability

In this experiment we assess OLED's scalability. When learning from the entire CAVIAR dataset (Section 6.2) the average processing time per training interpretation was 6.7 milliseconds (ms), while the frame rate in CAVIAR, i.e. the rate in which video frames containing new data arrive is 40 ms. As a "stress-test", we evaluated OLED's performance in more demanding learning tasks. We generated 4 different datasets, each of which consisted of a number of copies of CAVIAR. The new datasets differ from the original one in the constants referring to the tracked entities in simple and complex events. We generated datasets consisting of 2, 5, 8 and 10 copies, each of which contained 20, 50, 80 and 100 different entities respectively. Like in the previous experiments, each interpretation includes narrative and annotation atoms from two time points. In this experiment however, the number of atoms in each interpretation grows proportionally to the number of copies of the dataset.

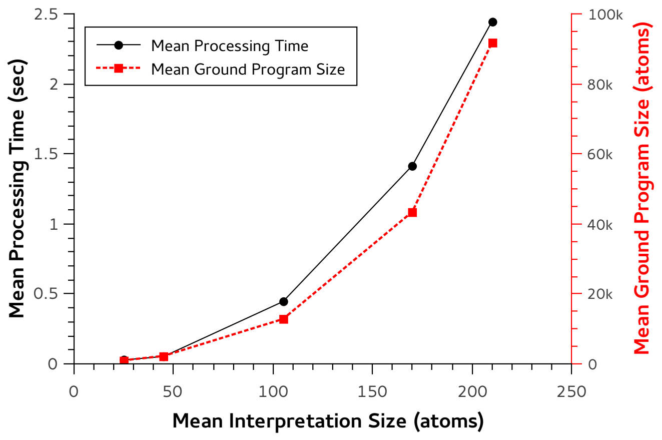

FIGURE 6.1: OLED's mean processing time and mean ground program size per training interpretation, for varying interpretation sizes.

We performed learning with OLED on the original and the enlarged datasets and measured the average processing time per training interpretation. Figure 6.1 presents the results. For instance, interpretations in the 10 copies of CAVIAR are handled in approximately 2.5 sec in a standard desktop computer. The growth in average processing time is due to the increased number of annotation atoms in the datasets, as well as the additional domain constants, that result in an exponential increase in the size of the ground program produced during the clause evaluation process (see the dashed line in Figure 6.1). OLED's performance may be improved by some optimizations, such as taking advantage of domain knowledge about relational dependencies in the data. For instance, in CAVIAR complex events involve two different entities, therefore learning may be split across different processing cores that learn from independent parts of the data. Such optimizations are part future work.

6.5 Summary

In this chapter we presented an experimental evaluation for OLED, using the CAVIAR dataset for activity recognition. We compared OLED to (a) a hand-crafted theory developed by domain experts; (b) a probabilistic version of this hand-crafted theory in the form of an MLN; (c) the XHAIL system and (d) the ILED system, presented in the previous chapter (Chapter 3). Our results indicate that OLED speeds-up the learning process by several orders of magnitude, as compared to XHAIL and the MLN-based approach of (b), while learning theories of comparable quality. Additionally, OLED outscores the hand-crafted theory of event definitions and is comparable to ILED on noise-free data fragments. We also assessed OLED's performance on a more demanding learning setting, containing (synthetic) training examples of larger sizes. We identified potential bottlenecks for in OLED's performance in such a learning setting and indicated directions for future work for resolving such bottlenecks and improving OLED's performance.

7 Conclusions and Future Work

In this thesis we focused on scalable relational learning for complex event recognition applications and presented two methods for learning complex event definitions in the form of domain-specific axioms in the Event Calculus. In Chapter 1 we discussed the basics of event recognition and we argued that symbolic event recognition systems based on first-order logic have significant advantages over non logic-based ones. These advantages include, among others, the ability for robust temporal reasoning via the incorporation of temporal action formalisms, such as the Event Calculus, and direct connections to machine learning tools for the automated extraction of event definitions from data with Inductive Logic Programming (ILP). In Chapter 2 we discussed the Event Calculus dialect that we used throughout this thesis and presented an overview of existing ILP methods for learning event definitions in the form of Event Calculus theories. We argued that the poor scalability of these methods is one of their main deficiencies, preventing them from being widely used in real-life applications. In subsequent chapters we developed two scalable approaches for learning Event Calculus theories with ILP and showed by means of experiments that both these approaches are efficient enough to be used in applications comprising large volumes of temporal data. In what follows we conclude this thesis by summarising the basic features of our proposed approaches and the respective experimental results, while we also discuss some directions for future work.

7.1 Conclusions

In Chapter 3 we presented the ILED system for Incremental Learning of Event Definitions. ILED is based on the XHAIL system, a state-of-art ILP learner that is capable of learning Event Calculus theories via a combination of abductive and inductive logic programming. ILED scales up the core XHAIL methodology, by adapting it so that it works with training examples arriving over time. Training examples are presented in the form of data chunks, each chunk consisting of examples found in a temporal window, whose size may be defined by the user. ILED works by progressively revising an initial hypothesis in the face of new examples, while adopting a full-memory approach, in which revisions must account for the entirety of the accumulated experience. By means of the support set, a compressive memory structure that encodes the coverage of clauses in the running hypothesis w.r.t. all past examples, ILED learns a sound hypothesis with no more than $2n$ revisions, where n is the number of data chunks. Moreover, thanks to the controlled size of training example chunks, the unit cost of each such revision is kept low, depending only on the size of each chunk, in terms of the number of domain constants in it. In Chapter 3 we presented ILED in detail and proved its soundness and the upper bound in the number of revisions required to compute a hypothesis. We also discussed related work in theory revision and identified the limitations of existing theory revision systems w.r.t. the problem that ILED addresses.

In Chapter 4 we presented an experimental evaluation for ILED, using real and synthetic data from a human activity recognition and a transport management application. Our results show that ILED scales adequately and is significantly more efficient than XHAIL, without compromising the quality of the learning outcome.

In Chapter 5 we presented OLED, an ILP system for learning event definitions in the form of Event Calculus theories in an online fashion. OLED differs from ILED in two key respects: First, it is able to handle noise in the training data, by learning imperfect hypotheses that cover large numbers of positive examples, while also potentially allowing to cover some negative examples. In contrast, ILED is designed for learning sound hypotheses. Second, OLED can learn from streams, i.e. continuous data flows that arrive at a high velocity. This learning setting is often desirable in event recognition applications, where the volume of the data makes their storage impractical and requires methods that build models with a single pass over the training stream. OLED incorporates a heuristic search, based on the Hoeffding bound, to learn clauses in an online fashion, using small subsets of the training stream to evaluate candidate clauses. This is done by relating the size of the subsets to a user-defined confidence level on the error margin of not selecting a (globally) optimal clause at each point during learning. OLED adapts a standard hill-climbing ILP search to make it work in an online fashion. The resulting search strategy requires learning each clause in isolation. This is hard to achieve when learning Event Calculus theories, where candidate clauses depend on each other via the core domain-independent axioms of the formalism. To overcome this issue OLED splits the learning process into two sub-processes, each of which is responsible for learning respectively the initiation and the termination part of the theory. This decoupling of the two types of clause allows to evaluate each clause independently, based on a scoring function that takes into account the potential utility of each individual clause in a theory.

In Chapter 6 we presented an experimental evaluation for OLED, using the CAVIAR dataset for activity recognition. We compared OLED to a hand-crafted theory, a probabilistic version of the latter in the form of a Markov Logic Network (MLN), the XHAIL system and the ILED system. The results show that OLED speeds up the learning process by several orders of magnitude, as compared to XHAIL and the MLN-based approach, while learning theories of comparable quality. Additionally, OLED outscores the hand-crafted theory of event definitions and is comparable to ILED on noise-free data fragments.

7.2 Future Work

There are several directions in which we intend to extend the work presented in this thesis. The main ones are outlined below:

Distributed Learning for Event Recognition. The main goal of this research direction is to further improve scalability of the learning systems presented in this work, via parallelizing parts of the learning task and distributing the workload over multiple processing cores. Some degree of parallelization exists already in the methods that we presented in this thesis (e.g. OLED learns initiation and termination clauses separately and in parallel). Moreover, we have already identified in the thesis some promising directions for further improving the performance of our proposed methods, by e.g. parallelizing the clause evaluation task in OLED, or utilizing expert knowledge about a particular domain, in order to partition the training set into independent data fragments and learn from each such fragment separately in a data-parallel fashion. As future work we intent to explore a variety of generic approaches for parallelizing relational learning tasks with ILP, including parallel exploration of independent hypotheses [Ohwada and Mizoguchi, 1999], parallelizing coverage tests for candidate clauses [Graham et al., 2003; Wang and Skillicorn, 2000], parallel execution of the core learning process over partitions of the training data [Dehaspe and De Raedt, 1995; Matsui et al., 1998] and so on. We intend to adjust such generic approaches to the particular tasks of incremental and online learning of event definitions proposed in this work.

Handling Noise and Uncertainty with Statistical Relational Learning. The OLED system, presented in Chapter 5, is capable of handling noise in the training data by learning imperfect hypotheses with a good fit in the data. Alternative approaches from the growing field of Statistical Relational Learning (SRL) [Getoor, 2007] seem to achieve robust performance w.r.t. to noise and uncertainty handling, via a combination of first-order logic and probability. SRL techniques have been combined with the Event Calculus in event recognition applications [Skarlatidis et al., 2015] and they have been used for learning both the structure and the parameters in such applications [Michelioudakis et al., 2016]. We are currently performing preliminary experiments that combine OLED with SRL techniques. In these experiments we use the MaxMargin parameter learning algorithm for learning the weights of a Markov Logic Network, whose structure is a direct translation, in Markov Logic syntax, of a set of clauses learnt by OLED. In future work we intent to extent the core ILED and OLED methodology in order to device algorithms for learning structure and parameters in an incremental and online fashion respectively.

Bibliography

Adé, H., De Raedt, L., and Bruynooghe, M. (1993). Theory revision. In *Proceedings of the 3rd International Workshop on Inductive Logic Programming*, pages 179–192.

Ade, H. and Denecker, M. (1995). AILP: Abductive inductive logic programming. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Ade, H., Malfait, B., and De Raedt, L. (1994). Ruth: an ILP theory revision system. In *International Symposium on Methodologies for Intelligent Systems (ISMIS)*, pages 336–345.

Adi, A. and Etzion, O. (2004). Amit-the situation manager. *The VLDB Journal-The International Journal on Very Large Data Bases*, 13(2):177–203.

Aggarwal, C. C. (2007). *Data streams: models and algorithms*, volume 31. Springer Science & Business Media.

Aggarwal, C. C. (2015). *Data mining: the textbook*. Springer.

Akman, V., Erdoğan, S. T., Lee, J., Lifschitz, V., and Turner, H. (2004). Representing the zoo world and the traffic world in the language of the causal calculator. *Artificial Intelligence*, 153(1):105–140.

Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: the sciff framework. *ACM Transactions on Computational Logic (TOCL)*, 9(4):29.

Alrajeh, D., Kramer, J., Russo, A., and Uchitel, S. (2011). An inductive approach for modal transition system refinement. In *Technical Communications of the International Conference of Logic Programming ICLP*, pages 106–116. Citeseer.

Alrajeh, D., Ray, O., Russo, A., and Uchitel, S. (2006). Extracting requirements from scenarios with ILP. In *Inductive Logic Programming*.

Alrajeh, D., Ray, O., Russo, A., and Uchitel, S. (2009). Using abduction and induction for operational requirements elaboration. *Journal of Applied Logic*, 7(3):275–288.

Anicic, D., Rudolph, S., Fodor, P., and Stojanovic, N. (2012). Stream reasoning and complex event processing in etalis. *Semantic Web*, 3(4):397–407.

Artikis, A., Paliouras, G., Portet, F., and Skarlatidis, A. (2010a). Logic-based representation, reasoning and machine learning for event recognition. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 282–293. ACM.

Artikis, A., Sergot, M., and Paliouras, G. (2015a). An event calculus for event recognition. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 27(4):895–908.

Artikis, A., Sergot, M. J., and Paliouras, G. (2015b). An event calculus for event recognition. *IEEE Trans. Knowl. Data Eng.*, 27(4):895–908.

Artikis, A., Skarlatidis, A., and Paliouras, G. (2010b). Behaviour recognition from video content: A logic programming approach. *International Journal on Artificial Intelligence Tools*, 19(2):193–209.

Artikis, A., Skarlatidis, A., and Paliouras, G. (2010c). Behaviour recognition from video content: a logic programming approach. *International Journal on Artificial Intelligence Tools*, 19(02):193–209.

Artikis, A., Skarlatidis, A., Portet, F., and Paliouras, G. (2012). Logic-based event recognition. *Knowledge Eng. Review*, 27(4):469–506.

Artikis, A., Weidlich, M., Schnitzler, F., Boutsis, I., Liebig, T., Piatkowski, N., Bockermann, C., Morik, K., Kalogeraki, V., Marecek, J., et al. (2014). Heterogeneous stream processing and crowdsourcing for urban traffic management. In *EDBT*, pages 712–723.

Athakravi, D., Corapi, D., Broda, K., and Russo, A. (2013). Learning through hypothesis refinement using answer set programming. In *Proc. of the 23rd Int. Conference of Inductive Logic Programming (ILP)*.

Badea, L. (2000). Learning trading rules with inductive logic programming. In *European Conference on Machine Learning*, pages 39–46. Springer.

Badea, L. (2001). A refinement operator for theories. In *Proc. of the Int. Conf. on Inductive Logic Programming (ILP)*.

Bain, M. and Muggleton, S. (1990). *Non-monotonic learning*. Citeseer.

Baral, C. (2003). *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press.

Bennett, G. (1962). Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45.

Bergadano, F., Gunetti, D., Nicosia, M., Ruffo, G., et al. (1996). Learning logic programs with negation as failure. *Advances in inductive logic programming*, pages 107–123.

Biba, M., Basile, T. M. A., Ferilli, S., and Esposito, F. (2006a). Improving scalability in ILP incremental systems. In *Proceedings of CILC 2006-Italian Conference on Computational Logic, Bari, Italy, June*, pages 26–27.

Biba, M., Basile, T. M. A., Ferilli, S., and Esposito, F. (2006b). Improving scalability in ILP incremental systems. In *Proc. of CILC 2006-Italian Conf. on Computational Logic*.

Blockeel, H. and De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1):285–297.

Blockeel, H., De Raedt, L., Jacobs, N., and Demoen, B. (1999). Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. (1990). Occam's razor. *Readings in machine learning*, pages 201–204.

Bragaglia, S. and Ray, O. (2014). Nonmonotonic learning in large biological networks. In *Proc. of the Int. Conf. on Inductive Logic Programming (ILP)*.

Bragaglia, S. and Ray, O. (2015). Nonmonotonic learning in large biological networks. In *Inductive Logic Programming*, pages 33–48. Springer.

Bratko, I. (1999). Refining complete hypotheses in ilp. In *International Conference on Inductive Logic Programming*, pages 44–55. Springer.

Bratko, I. (2001). *Prolog programming for artificial intelligence*. Pearson education.

Brendel, W., Fern, A., and Todorovic, S. (2011). Probabilistic event logic for interval-based event recognition. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3329–3336. IEEE.

Broda, K., Clark, K., Miller, R., and Russo, A. (2009). Sage: a logical agent-based environment monitoring and control system. In *European Conference on Ambient Intelligence*, pages 112–117. Springer.

Callens, L., Carrault, G., Cordier, M. O., Fromont, E., Portet, F., and Quiniou, R. (2008a). Intelligent adaptive monitoring for cardiac surveillance. *European Conference on Artificial Intelligence (ECAI)*, pages 653–657.

Callens, L., Carrault, G., Cordier, M.-O., Fromont, E., Portet, F., and Quiniou, R. (2008b). Intelligent adaptive monitoring for cardiac surveillance. In *European Conference on Artificial Intelligence*, pages 653–657.

Calzone, L., Chabrier-Rivier, N., Fages, F., and Soliman, S. (2006). Machine learning biochemical networks from temporal logic properties. In *Transactions on Computational Systems Biology VI*, pages 68–94. Springer.

Carrault, G., Cordier, M.-O., Quiniou, R., and Wang, F. (2003). Temporal abstraction and inductive logic programming for arrhythmia recognition from electrocardiograms. *Artificial intelligence in medicine*, 28(3):231–263.

Cattafi, M., Lamma, E., Riguzzi, F., and Storari, S. (2010). Incremental declarative process mining. *Smart Information and Knowledge Management*, pages 103–127.

Cervesato, I. and Montanari, A. (2000). A calculus of macro-events: Progress report. In *Proc. of the Int. Workshop on Temporal Representation and Reasoning (TIME)*. IEEE.

Chaudet, H. (2006). Extending the event calculus for tracking epidemic spread. *Artificial Intelligence in Medicine*, 38(2):137–156.

Chaudhry, N., Shaw, K., and Abdelguerfi, M. (2006). *Stream data management*, volume 30. Springer Science & Business Media.

Chernoff, H. (1952). A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507.

Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., and Storari, S. (2009). Exploiting inductive logic programming techniques for declarative process mining. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 278–295. Springer.

Chittaro, L. and Dojat, M. (1997). Using a general theory of time and change in patient monitoring: experiment and evaluation. *Computers in Biology and Medicine*, 27(5):435–452.

Choppy, C., Bertrand, O., and Carle, P. (2009). Coloured petri nets for chronicle recognition. In *International Conference on Reliable Software Technologies*, pages 266–281. Springer.

Clark, K. L. (1977). Negation as failure. In *Logic and Data Bases*, pages 293–322.

Clark, K. L. (1978). Negation as failure. In *Logic and data bases*, pages 293–322. Springer.

Clarke, E. M., Grumberg, O., and Peled, D. (1999). *Model checking*. MIT press.

Corapi, D. (2012). *Nonmonotonic inductive logic programming as abductive search*. PhD thesis, Imperial College London.

Corapi, D., De Vos, M., Padget, J., Russo, A., and Satoh, K. (2011a). Norm refinement and design through inductive learning. In *Coordination, Organizations, Institutions, and Norms in Agent Systems VI*, pages 77–94. Springer.

Corapi, D., Ray, O., Russo, A., Bandara, A., and Lupu, E. (2008). Learning rules from user behaviour. In *Second International Workshop on the Induction of Process Models*.

Corapi, D., Russo, A., De Vos, M., Padget, J., and Satoh, K. (2011b). Normative design using inductive learning. *Theory and Practice of Logic Programming*, 11(4-5):783–799.

Corapi, D., Russo, A., and Lupu, E. (2010). Inductive logic programming as abductive search. In *Technical Communications of the Int. Conf. on Logic Programming (ICLP)*.

Corapi, D., Russo, A., and Lupu, E. (2012). Inductive logic programming in answer set programming. In *Proc. of Int. Conf. on Inductive Logic Programming (ILP)*. Springer.

Cormode, G., Muthukrishnan, S., and Zhuang, W. (2007). Conquering the divide: Continuous clustering of distributed data streams. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1036–1045. IEEE.

Cugola, G. and Margara, A. (2010). Tesla: a formally defined event specification language. In *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*, pages 50–61. ACM.

Cugola, G. and Margara, A. (2012). Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62.

D. Raedt, L. (1992). *Interactive theory revision: an inductive logic programming approach*. Academic Press Ltd.

Dardenne, A., Van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50.

De Raedt, L. (1997). Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201.

De Raedt, L. (2008). *Logical and relational learning*. Springer Science & Business Media.

De Raedt, L. and Bruynooghe, M. (1994). Interactive theory revision. In *Machine Learning: a Multistrategy Approach*, pages 239–263.

De Raedt, L. and Džeroski, S. (1994). First-order jk-clausal theories are pac-learnable. *Artificial Intelligence*, 70(1):375–392.

De Raedt, L. and Van Laer, W. (1995). Inductive constraint logic. In *International Workshop on Algorithmic Learning Theory*, pages 80–94. Springer.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95.

Dehaspe, L. and De Raedt, L. (1995). Parallel inductive logic programming. In *Proceedings of the MLnet familiarization workshop on statistics, machine learning and knowledge discovery in databases*, volume 1, page 5. Citeseer.

Demolombe, R., del Cerro, L. F., and Obeid, N. (2013). A logical model for metabolic networks with inhibition. In *Proceedings of the International Conference on Bioinformatics & Computational Biology (BIOCOMP)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

Denecker, M. and Kakas, A. (2002). Abduction in logic programming. In *Computational Logic: Logic Programming and Beyond*, pages 402–436.

Dhurandhar, A. and Dobra, A. (2010). Test set bounds for relational data that vary with strength of dependence. *submitted to ACM Transactions on Computational Logic*.

Dhurandhar, A. and Dobra, A. (2012). Distribution-free bounds for relational classification. *Knowl. Inf. Syst.*, 31(1):55–78.

Di Mauro, N., Esposito, F., Ferilli, S., and Basile, T. M. (2005). Avoiding order effects in incremental learning. In *AIIA 2005: Advances in Artificial Intelligence*, pages 110–121.

Di Mauro, N., Esposito, F., Ferilli, S., and Basile, T. M. A. (2004). A backtracking strategy for order-independent incremental learning. In *Proc. of the European Conf. on Artificial Intelligence (ECAI)*.

Dimopoulos, Y. and Kakas, A. (1995). Learning non-monotonic logic programs: Learning exceptions. In *European Conference on Machine Learning*, pages 122–137. Springer.

Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnström, J. (1998). Tal: Temporal action logics language specification and tutorial.

Domingos, P. and Hulten, G. (2001). A general method for scaling up machine learning algorithms and its application to clustering. In *ICML*, volume 1, pages 106–113.

Domingos, P. M. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 71–80.

Doncescu, A., Inoue, K., and Yamamoto, Y. (2007). Knowledge based discovery in systems biology using cf-induction. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 395–404. Springer.

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

Dubba, K., Bhatt, M., Dylla, F., Hogg, D. C., and Cohn, A. G. (2011). Interleaved inductive-abductive reasoning for learning complex event models. In *International Conference on Inductive Logic Programming*, pages 113–129. Springer.

Dubba, K. S. R., Cohn, A. G., and Hogg, D. C. (2010). Event model learning from complex videos using ilp. In *ECAI*, volume 215, pages 93–98.

Dubba, K. S. R., Cohn, A. G., Hogg, D. C., Bhatt, M., and Dylla, F. (2015). Learning relational event models from video. *Journal of Artificial Intelligence Research*, 53:41–90.

Duboc, A. L., Paes, A., and Zaverucha, G. (2009). Using the bottom clause and mode declarations in FOL theory revision from examples. *Machine Learning*, 76(1):73–107.

Eckert, M. and Bry, F. (2010). Rule-based composite event queries: the language xchangeeq and its semantics. *Knowledge and information systems*, 25(3):551–573.

Eshghi, K. and Kowalski, R. (1989). Abduction compared with negation by failure. In *Proceedings of the 6th International Conference on Logic Programming*.

Esposito, F., Ferilli, S., Fanizzi, N., Basile, T. M. A., and Di Mauro, N. (2004). Incremental learning and concept drift in inthelex. *Intelligent Data Analysis*, 8(3):213–237.

Esposito, F., Laterza, A., Malerba, D., and Semeraro, G. (1996). Refinement of datalog programs. In *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming*.

Esposito, F., Semeraro, G., Fanizzi, N., and Ferilli, S. (2000). Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 28(1-2):133–156.

Etzion, O. and Niblett, P. (2010). *Event processing in action*. Manning Publications Co.

Fages, F. and Soliman, S. (2008). Model revision from temporal logic properties in computational systems biology. In *Probabilistic inductive logic programming*, pages 287–304. Springer.

Flach, P. A. (1998). The logic of learning: a brief introduction to inductive logic programming. In *Proceedings of the CompulogNet Area Meeting on Computational Logic and Machine Learning*, pages 1–17.

Floyd, S. and Warmuth, M. (1995). Sample compression, learnability, and the vapnik-chervonenkis dimension. *Machine learning*, 21(3):269–304.

Fogel, L. and Zaverucha, G. (1998). Normal programs and multiple predicate learning. In *Inductive Logic Programming*, pages 175–184. Springer.

Gaber, M. M., Gama, J., Krishnaswamy, S., Gomes, J. B., and Stahl, F. (2014). Data stream mining in ubiquitous environments: state-of-the-art and current directions. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(2):116–138.

Gama, J. (2010). *Knowledge discovery from data streams*. CRC Press.

Gama, J. and Gaber, M. M. (2007). *Learning from data streams*. Springer.

Gama, J., Kosina, P., et al. (2011). Learning decision rules from data streams. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1255. Citeseer.

Gebser, M., Kaminski, R., Kaufmann, B., and Schaub, T. (2012). Answer set solving in practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(3):1–238.

Gelfond, M. (2008). Answer sets. In van Harmelen, Frank; Lifschitz, V. P. B., editor, *Handbook of Knowledge Representation*. Elsevier.

Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *International Conference on Logic Programming*, pages 1070–1080.

Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *The Journal of Logic Programming*, 17(2):301–321.

Gelfond, M. and Lifschitz, V. (1998). Action languages.

Getoor, L. (2007). *Introduction to statistical relational learning*. MIT press.

Ghallab, M. (1996). On chronicles: Representation, on-line recognition and learning. In *KR*, pages 597–606.

Giraud-Carrier, C. (2000). A note on the utility of incremental learning. *AI Communications*, 13(4).

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, 153(1):49–104.

G.Plotkin (1970). A note on inductive generalization. *Machine Intelligence*, 5:153–163.

Graham, J., Page, C. D., and Kamal, A. (2003). Accelerating the drug design process through parallel inductive logic programming data mining. In *Proceedings of the IEEE Computer Society Conference on Bioinformatics*, page 400. IEEE Computer Society.

Hirata, K. (1999). Flattening and implication. In *International Conference on Algorithmic Learning Theory*, pages 157–168. Springer.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.

Hulten, G., Domingos, P., and Abe, Y. (2003). Mining massive relational databases. In *Proceedings of the IJCAI-2003 workshop on learning statistical models from relational data*, pages 53–60.

Hulten, G., Domingos, P., and Spencer, L. (2005). Mining massive data streams. *The Journal of Machine Learning Research*.

Hulten, G., Spencer, L., and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM.

Huynh, T. N. and Mooney, R. J. (2009). Max-margin weight learning for markov logic networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 564–579. Springer.

Ikonomovska, E., Gama, J., and Džeroski, S. (2011). Learning model trees from evolving data streams. *Data mining and knowledge discovery*, 23(1):128–168.

Inoue, K. (2004). Induction as consequence finding. *Machine Learning*, 55(2):109–135.

Inoue, K., Bando, H., and Nabeshima, H. (2005). Inducing causal laws by regular inference. In *International Conference on Inductive Logic Programming*, pages 154–171. Springer.

Inoue, K., Doncescu, A., and Nabeshima, H. (2013). Completing causal networks by meta-level abduction. *Machine learning*, 91(2):239–277.

Inoue, K., Ribeiro, T., and Sakama, C. (2014). Learning from interpretation transition. *Machine Learning*, 94(1):51–79.

Janiesch, C., Matzner, M., and Müller, O. (2011). A blueprint for event-driven business activity management. In *International Conference on Business Process Management*, pages 17–28. Springer.

Japkowicz, N. and Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press.

Jensen, D. (1999). Statistical challenges to inductive inference in linked data. In *AISTATS*.

Jensen, D. and Neville, J. (2002). Autocorrelation and linkage cause bias in evaluation of relational learners. In *Inductive Logic Programming*, pages 101–116. Springer.

Kakas, A., Kowalski, R., and Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, 2:719–770.

Kakas, A. and Mancarella, P. (1990). Generalised stable models: A semantics for abduction. In *ninth European Conference on Artificial Intelligence (ECAI-90)*, pages 385–391.

Katzouris, N., Artikis, A., and Paliouras, G. (2014). Event recognition for unobtrusive assisted living. In *Hellenic Conference on Artificial Intelligence*, pages 475–488. Springer.

Katzouris, N., Artikis, A., and Paliouras, G. (2015). Incremental learning of event definitions with inductive logic programming. *Machine Learning*, 100(2-3):555–585.

Katzouris, N., Artikis, A., and Paliouras, G. (2016). Online learning of event definitions. *TPLP*, 16(5-6):817–833.

Kimber, T. (2012). *Learning definite and normal logic programs by induction on failure*. PhD thesis, Imperial College London.

Kimber, T., Broda, K., and Russo, A. (2009). Induction on failure: Learning connected horn theories. In *Logic Programming and Nonmonotonic Reasoning*, pages 169–181.

Könik, T. and Laird, J. E. (2006). Learning goal hierarchies from structured observations and expert annotations. *Machine Learning*, 64(1-3):263–287.

Kosina, P. and Gama, J. (2012). Handling time changing data with adaptive very fast decision rules. In *Machine Learning and Knowledge Discovery in Databases*, pages 827–842. Springer.

Kowalski, R. and Sergot, M. (1986a). A logic-based calculus of events. *New Generation Computing*, 4(1):67–96.

Kowalski, R. A. and Sergot, M. J. (1986b). A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95.

Kuzelka, O. and Zelezny, F. (2008). A restarted strategy for efficient subsumption testing. *Fundamenta Informaticae*, 89(1).

Laguna, J. O. (2014). *Building Planning Action Models Using Activity Recognition*. PhD thesis, Universoidad Carlos III De Madrid.

Lamma, E., Mello, P., Riguzzi, F., and Storari, S. (2007). Applying inductive logic programming to process mining. In *International Conference on Inductive Logic Programming*, pages 132–146. Springer.

Langford, J. (2005). Tutorial on practical prediction theory for classification. *Journal of machine learning research*, 6(Mar):273–306.

Langley, P. (1995). *Learning in Humans and Machines: Towards an Interdisciplinary Science*, chapter Order Effects in Incremental Learning. Elsevier.

Lavrač, N. and Džeroski, S. (1993). *Inductive Logic Programming: Techniques and Applications*. Routledge.

Law, M., Russo, A., and Broda, K. (2014). Inductive learning of answer set programs. In *European Workshop on Logics in Artificial Intelligence*, pages 311–325. Springer.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge University Press.

Li, H.-F. and Lee, S.-Y. (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2):1466–1477.

Li, H.-F., Lee, S.-Y., and Shan, M.-K. (2004). An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proc. of First International Workshop on Knowledge Discovery in Data Streams*.

List, T., Bins, J., Vazquez, J., and Fisher, R. B. (2005). Performance evaluating the evaluator. In *2nd Joint IEEE Int. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 129–136. IEEE.

Lloyd, J. (1987). *Foundations of Logic Programming*. Springer.

Lopes, C. and Zaverucha, G. (2009). Htilde: scaling up relational decision trees for very large databases. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1475–1479. ACM.

Lorenzo, D. (2002). Learning non-monotonic causal theories from narratives of actions. In *NMR*, pages 349–355.

Lorenzo, D. and Otero, R. P. (2000). Learning to reason about actions. In *ECAI*, pages 316–320.

Luckham, D. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc.

Maggi, F. M., Corapi, D., Russo, A., Lupu, E., and Visaggio, G. (2011). Revising process models through inductive learning. In *Business Process Management Workshops*, pages 182–193. Springer.

Maloberti, J. and Sebag, M. (2004). Fast theta-subsumption with constraint satisfaction algorithms. *Machine Learning*, 55(2):137–174.

Maloof, M. A. and Michalski, R. S. (2004). Incremental learning with partial instance memory. *Artificial intelligence*, 154(1):95–126.

Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment.

Margara, A., Cugola, G., and Tamburrelli, G. (2014). Learning from the past: automated rule generation for complex event processing. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 47–58. ACM.

Margara, A., Cugola, G., Tamburrelli, G., and Lugano, I. (2013). Towards automated rule learning for complex event processing. Technical report, Technical Report.

Martin, L. and Vrain, C. (1996). A three-valued framework for the induction of general logic programs. *Advances in Inductive Logic Programming*, pages 219–235.

Matsui, T., Inuzuka, N., Seki, H., and Itoh, H. (1998). Comparison of three parallel implementations of an induction algorithm. In *8th Int. Parallel Computing Workshop*, pages 181–188. Citeseer.

McAllester, D. A. (1999). Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170. ACM.

McCarthy, J. (1986). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28(1):89–116.

McCarthy, J. (2002). Actions and other events in situation calculus. In *KR*, pages 615–628.

Michelioudakis, E., Skarlatidis, A., Paliouras, G., and Artikis, A. (2016). Osla: Online structure learning using background knowledge axiomatization.

Miller, R. and Shanahan, M. (2002). Some alternative formulations of the event calculus. In *Computational logic: logic programming and beyond*, pages 452–490. Springer.

Mitchell, T. (1979). *Version Spaces: An Approach to Concept Learning.* PhD thesis. AAI7917262.

Mooney, R. J. (1992). *Batch versus incremental theory refinement*. Artificial Intelligence Laboratory, University of Texas at Austin.

Motwani, R. and Raghavan, P. (2010). *Randomized algorithms*. Chapman & Hall/CRC.

Moyle, S. (2002). Using theory completion to learn a robot navigation control program. In *International Conference on Inductive Logic Programming*, pages 182–197. Springer.

Moyle, S. and Muggleton, S. (1997). Learning programs in the event calculus. *7th International Workshop on Inductive Logic Programming*, 1297:205–212.

Moyle, S. A. (2003). *An investigation into theory completion techniques in inductive logic programming*. PhD thesis, University of Oxford.

Mueller, E. T. (2008). The event calculus. In van Harmelen, Frank; Lifschitz, V. P. B., editor, *Handbook of Knowledge Representation*. Elsevier.

Mueller, E. T. (2014). *Commonsense reasoning: An event calculus based approach*. Morgan Kaufmann.

Muggleton, S. (1995a). Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286.

Muggleton, S. (1995b). Inverse entailment and progol. *New generation computing*, 13(3-4):245–286.

Muggleton, S. and Bryant, C. (2000a). Theory completion using inverse entailment. In *International Conference on Inductive Logic Programming*, pages 130–146.

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.

Muggleton, S., Feng, C., et al. (1990). *Efficient induction of logic programs*. Citeseer.

Muggleton, S., Paes, A., Costa, V. S., and Zaverucha, G. (2012). Chess revision: acquiring the rules of chess variants through fol theory revision from examples. In *Inductive Logic Programming*.

Muggleton, S. H. and Bryant, C. H. (2000b). Theory completion using inverse entailment. In *International conference on inductive logic programming*, pages 130–146. Springer.

Muggleton, S. H., Lin, D., Pahlavi, N., and Tamaddoni-Nezhad, A. (2014). Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94(1):25–49.

Muggleton, S. H., Santos, J. C. A., and Tamaddoni-Nezhad, A. (2008). Toplog: Ilp using a logic program declarative bias. In *International Conference on Logic Programming*, pages 687–692. Springer.

Muthukrishnan, S. (2005). *Data streams: Algorithms and applications*. Now Publishers Inc.

Nabeshima, H., Iwanuma, K., and Inoue, K. (2003). Solar: a consequence finding system for advanced reasoning. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 257–263. Springer.

Needham, C. J., Santos, P. E., Magee, D. R., Devin, V., Hogg, D. C., and Cohn, A. G. (2005). Protocols from perceptual observations. *Artificial Intelligence*, 167(1):103–136.

Nicolas, P. and Duval, B. (2001). Representation of incomplete knowledge by induction of default theories. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 160–172. Springer.

Nienhuys-Cheng, S.-H. and De Wolf, R. (1997). *Foundations of inductive logic programming*, volume 1228. Springer Science & Business Media.

Ohwada, H. and Mizoguchi, F. (1999). Parallel execution for speeding up inductive logic programming systems. In *International Conference on Discovery Science*, pages 277–286. Springer.

Otero, R. P. (2001). Induction of stable models. In *Inductive Logic Programming*, pages 193–205. Springer.

Otero, R. P. (2003). Induction of the effects of actions by monotonic methods. In *Inductive Logic Programming*, pages 299–310. Springer.

Otero, R. P. (2004). Embracing causality in inducing the effects of actions. In *Current Topics in Artificial Intelligence*, pages 291–301. Springer.

Paes, A., Zaverucha, G., and Costa, V. S. (2007). Revising first-order logic theories from examples through stochastic local search. In *International Conference on Inductive Logic Programming*, pages 200–210. Springer.

Paschke, A. (2005). ECA-RuleML: An approach combining ECA rules with temporal interval-based KR event logics and transactional update logics. Technical report, Technische Universitat Munchen.

Paschke, A. (2006). Eca-ruleml: An approach combining eca rules with temporal interval-based kr event/action logics and transactional update logics. *arXiv preprint cs/0610167*.

Paschke, A. and Bichler, M. (2008). Knowledge representation concepts for automated sla management. *Decision Support Systems*, 46(1):187–205.

Paschke, A. and Kozlenkov, A. (2009). Rule-based event processing and reaction rules. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 53–66. Springer.

Paschke, A., Kozlenkov, A., and Boley, H. (2010). A homogeneous reaction rule language for complex event processing. *arXiv preprint arXiv:1008.0823*.

Patroumpas, K., Alevizos, E., Artikis, A., Vodas, M., Pelekis, N., and Theodoridis, Y. (2016). Online event recognition from moving vessel trajectories. *arXiv preprint arXiv:1601.06041*.

Patroumpas, K., Artikis, A., Katzouris, N., Vodas, M., Theodoridis, Y., and Pelekis, N. (2015). Event recognition for maritime surveillance. In *EDBT*, pages 629–640.

Plotkin, G. D. (1970). A note on inductive generalization. *Machine intelligence*, 5(1):153–163.

Pnueli, A. and Manna, Z. (1992). The temporal logic of reactive and concurrent systems.

Quinlan, J. R. (1990a). Learning logical definitions from relations. *Machine Learning*, 5:239–266.

Quinlan, J. R. (1990b). Learning logical definitions from relations. *Machine learning*, 5(3):239–266.

Randell, D. A., Cui, Z., and Cohn, A. G. (1992). A spatial logic based on regions and connection. *KR*, 92:165–176.

Ray, O. (2005). Hybrid abductive inductive learning. *PhD thesis*, Department of Computing, Imperial College London, UK.

Ray, O. (2006). Using abduction for induction of normal logic programs. In *ECAI'06 Workshop on Abduction and Induction in Artificial Intelligence and Scientific Modelling*.

Ray, O. (2009a). Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340.

Ray, O. (2009b). Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340.

Ray, O., Broda, K., and Russo, A. (2003). Hybrid abductive inductive learning: A generalisation of progol. In *Proc. of the Int. Conf. in Inductive Logic Programming (ILP)*.

Ray, O., Broda, K., and Russo, A. (2004). Generalised kernel sets for inverse entailment. In *International Conference in Logic Programming (ICPL)*, pages 165–179.

Ray, O. and Bryant, C. H. (2008). Inferring the function of genes from synthetic lethal mutations. In *Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. International Conference on*, pages 667–671. IEEE.

Ray, O. and Inoue, K. (2007). Mode-directed inverse entailment for full clausal theories. In *International Conference on Inductive Logic Programming*, pages 225–238. Springer.

Ray, O., Whelan, K., and King, R. (2010). Logic-based steady-state analysis and revision of metabolic networks with inhibition. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 661–666. IEEE.

Richards, B. and Mooney, R. (1995). Automated refinement of first-order horn clause domain theories. *Machine Learning*, 19(2):95–131.

Richards, B. L. and Mooney, R. J. (1991). First order theory revision. In *8th International Workshop on Machine Learning*, pages 447–451.

Rodrigues, C., Gérard, P., and Rouveirol, C. (2010a). Incremental learning of relational action models in noisy environments. In *International Conference on Inductive Logic Programming*, pages 206–213. Springer.

Rodrigues, C., Gérard, P., Rouveirol, C., and Soldano, H. (2010b). Incremental learning of relational action rules. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 451–458. IEEE.

Rodrigues, C., Gérard, P., Rouveirol, C., and Soldano, H. (2011). Active learning of relational action models. In *International Conference on Inductive Logic Programming*, pages 302–316. Springer.

Rodrigues, P. P., Gama, J., and Pedroso, J. (2008). Hierarchical clustering of time-series data streams. *IEEE transactions on knowledge and data engineering*, 20(5):615–627.

Rouveirol, C. (1994). Flattening and saturation: Two representation changes for generalization. *Machine Learning*, 14(2):219–232.

Saitta, L. (2010). *Ubiquitous Knowledge Discovery*. Springer.

Sakama, C. (1999). Some properties of inverse resolution in normal logic programs. In *International Conference on Inductive Logic Programming*, pages 279–290. Springer.

Sakama, C. (2000). Inverse entailment in nonmonotonic logic programs. In *Proc. of the Int. Conf. on Inductive Logic Programming (ILP)*.

Sakama, C. (2001). Nonmonotomic inductive logic programming. In *Logic Programming and Nonmotonic Reasoning*, pages 62–80. Springer.

Sakama, C. (2005). Induction from answer sets in nonmonotonic logic programs. *ACM Transactions on Computational Logic*, 6 (2):203–231.

Sakama, C. and Inoue, K. (2009). Brave induction: a logical framework for learning from incomplete information. *Machine Learning*, 76(1):3–35.

Sandewall, E. (1992). *Features and fluents: A systematic approach to the representation of knowledge about dynamical systems*. Linköping University.

Santos, J. and Muggleton, S. (2010). Subsumer: A prolog theta-subsumption engine. In *Technical Communications of the 26th Int. Conf. on Logic Programming*.

Schultz-Møller, N. P., Migliavacca, M., and Pietzuch, P. (2009). Distributed complex event processing with query rewriting. In *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, page 4. ACM.

Seitzer, J. (1997). Stable ilp: exploring the added expressivity of negation in the background knowledge. In *Proceedings of IJCAI-95 Workshop on Frontiers of ILP*, volume 76, page 77. Citeseer.

Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., and Ferilli, S. (1997). A logic framework for the incremental inductive synthesis of datalog theories. In *International Workshop on Logic Programming Synthesis and Transformation*, pages 300–321. Springer.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

Skarlatidis, A., Paliouras, G., Artikis, A., and Vouros, G. A. (2015). Probabilistic event calculus for event recognition. *ACM Transactions on Computational Logic (TOCL)*, 16(2):11.

Srinivasan, A. (2000). The aleph manual. *http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/*.

Storf, H., Kleinberger, T., Becker, M., Schmitt, M., Bomarius, F., and Prueckner, S. (2009). An event-driven approach to activity recognition in ambient assisted living. In *European Conference on Ambient Intelligence*, pages 123–132. Springer.

Synnaeve, G., Inoue, K., Doncescu, A., Nabeshima, H., Kameya, Y., Ishihata, M., and Sato, T. (2011). Kinetic models and qualitative abstraction for relational learning in systems biology. In *BIOSTEC Bioinformatics 2011*.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A., and Muggleton, S. (2006). Application of abductive ilp to learning metabolic network inhibition from temporal data. *Machine Learning*, 64(1-3):209–230.

Tamaddoni-Nezhad, A., Chaleil, R., Kakas, A. C., Sternberg, M., Nicholson, J., and Muggleton, S. (2007). Modeling the effects of toxins in metabolic networks. *IEEE Engineering in Medicine and Biology Magazine*, 26(2):37.

Tamaddoni-Nezhad, A., Kakas, A., Muggleton, S., and Pazos, F. (2004). Modelling inhibition in metabolic pathways through abduction and induction. In *International Conference on Inductive Logic Programming*, pages 305–322. Springer.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial intelligence*, 111(1):277–299.

Utgoff, P. E. (2011). Incremental learning. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of machine learning*. Springer Science & Business Media.

Van der Aalst, W. M., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., and Weijters, A. J. (2003). Workflow mining: a survey of issues and approaches. *Data & knowledge engineering*, 47(2):237–267.

Van Dongen, B. F., De Medeiros, A. A., and Wen, L. (2009). Process mining: Overview and outlook of petri net discovery algorithms. In *Transactions on Petri Nets and Other Models of Concurrency II*, pages 225–242. Springer.

Van Dongen, B. F. and Van der Aalst, W. M. (2004). Multi-phase process mining: Building instance graphs. In *International Conference on Conceptual Modeling*, pages 362–376. Springer.

Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

Vovk, V., Gammerman, A., and Shafer, G. (2005). *Algorithmic learning in a random world*. Springer Science & Business Media.

Wang, Y., Cao, K., and Zhang, X. (2013). Complex event processing over distributed probabilistic event streams. *Computers & Mathematics with Applications*, 66(10):1808–1821.

Wang, Y. and Skillicorn, D. (2000). Parallel inductive logic for data mining. In *Workshop on Distributed and Parallel Knowledge Discovery, KDD2000, Boston*. Citeseer.

Westendorp, J. (2002). Noise-resistant incremental relational learning using possible worlds. In *International Conference on Inductive Logic Programming*, pages 317–332. Springer.

Wogulis, J. and Pazzani, M. (1993). A methodology for evaluating theory revision systems: Results with audrey ii. In *13th Interantional Joint Conference in Artificial Intelligence IJCAI*, pages 1128–1134.

Wrobel, S. (1994). Concept formation during interactive theory revision. *Machine Learning*, 14(2):169–191.

Wrobel, S. (1996). First order theory refinement. In De Raedt, L., editor, *Advances in Inductive Logic Programming*, pages 14 – 33.

Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In *International Conference in Inductive Logic Programming (ILP)*, pages 296–308.

Yamamoto, A. (2000). Using abduction for induction based on bottom generalization. In *Abduction and Induction*, pages 267–280. Springer.

Yamamoto, A. (2003). Hypothesis finding based on upward refinement of residue hypotheses. *Theoretical Computer Science*, 298(1):5–19.

Yamamoto, Y., Inoue, K., and Doncescu, A. (2008). Estimation of possible reaction states in metabolic pathways using inductive logic programming. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 808–813. IEEE.

Yamamoto, Y., Inoue, K., and Doncescu, A. (2010). Integrating abduction and induction in biological inference using cf-induction. *Elements of computational systems biology*, pages 213–234.

Yang, H. and Fong, S. (2011). Moderated vfdt in stream mining using adaptive tie threshold and incremental pruning. In *Data Warehousing and Knowledge Discovery*, pages 471–483. Springer.