Pre-Proceedings of the 2007 International Workshop on Abduction and Induction in Artificial Intelligence (AIAI'07)

Abduction and induction are forms of logical reasoning that are used in many applications of Artificial Intelligence (AI). Individually, abduction reasons from effects to possible causes, while induction learns general rules for particular observations. In combination, abduction and induction can be integrated within an iterative cycle of knowledge development. But, recent attempts to such hybrid methods in tasks that require spatio-temporal learning and reasoning, have begun to reveal a need for more expressive formalisms and inference mechanisms than offered by existing systems.

This workshop will focus on expressive methodologies and applications of logic-based abduction and induction. To provide a concrete application domain, the AIAI'07 workshop will be co-located with the 1st Franco-Japanese Symposium on Knowledge Discovery in Systems Biology. The aims of the workshop are to investigate need for expressive logical formalisms in bioinformatics and other areas of AI, to idendity the state-of-the-art inference techniques for handling such formalisms, and to encourage interaction between biologists and logicians by discussing potential application areas and suggesting possible solutions.

The workshop will be held on the 15th of September 2007 at the Aquabella Hotel in the French town of Aix-en-Provence. The event is generously sponsored by the Laboratory for the Analysis and Architecture of Systems at the French National Centre for Scientific Research (LAAS-CNRS) and the Japanese National Institute of Informatics (NII) as part of a strategic Franco-Japanese collaboration on Knowledge-based Discovery in Systems Biology.

Andrei Doncescu
Peter A. Flach
Katsumi Inoue
Antonis C. Kakas
Oliver Ray

(Workshop Organisers)

September 2007

Programme Committee

Chitta Baral, Arizona State University, USA
Andrei Doncescu, National Center for Scientific Research, France
Peter Flach, University of Bristol, UK
Katsumi Inoue, National Institute of Informatics, Japan
Antonis Kakas, University of Cyprus, Cyprus
Ross King, University of Wales, UK
Ramon Otero, University of Corunna, Spain
Oliver Ray, University of Bristol, UK
Chiaki Sakama, Wakayama University, Japan

Workshop Participants

Dalal Alrajeh (Imperial College London, UK)
Chitta Baral (Arizona State University, USA)
Gauvain Bourgne (Univ, Paris IX, France)
Henning Christiansen (Roskilde University, Denmark)
Jacques Demongeot (TIMC-IMAG Laboratory, France)
Andrei Doncescu (LAAS-CNRS, France)
Louis Farinas del Cerro (IRIT, France)
Gerald Goma (INSA-INRA-CNRS, France)
Katsumi Inoue (National Institute of Informatics, Japan)
Koji Iwanuma (University of Yamanashi, Japan)
Carole Jouve (INSA-INRA-CNRS, France)
Antonis Kakas (University of Cyprus, Cyprus)
Yoshitaka Kameya (Tokyo Institute or Technology, Japan)
Nicolas Maudet (Univ. Paris IX, France)
Emmanuel Montseny (LAAS-CNRS, France)
Gerard Montseny (LAAS-CNRS, France)
Hidetomo Nabeshima (University of Yamanashi, Japan)
Ramon Otero (University of Corunna, Spain)
Oliver Ray (University of Bristol, UK)
Gilles Richard (IRIT, France)
Gilles Roux (LAAS-CNRS, France)
Alessandra Russo (Imperial College London, UK)
Chiaki Sakama (Wakayama University, Japan)
Taisuke Sato (Tokyo Institute of Technology, Japan)
Pierre Siegel (Univ. Provence, France)
Louise Trave-Massuyes (LAAS-CNRS, France)
Jean Louis Urribelarea (INSA-INRA-CNRS, France)
Yoshitaka Yamamoto (The Graduate University for Advanced Studies, Japan)

Programme

09.15 OPEN

09.30 Invited Talk: To be announced
Chitta Baral

10.15 Logic-statistic modeling and analysis of biological sequence data: A Research Agenda
Henning Christiansen

10.45 COFFEE

11.15 Invited Talk: To be announced
Ramon Otero

12.00 Nonmonotonic Abductive Inductive Learning
Oliver Ray

12.30 LUNCH

14.15 Towards Refinement of Abductive or Inductive Hypothesis through Propagation
Gauvain Bourgne, Amal El Fallah Seghrouchni and Nicolas Maudet

14.45 Reconsideration of Circumscriptive Induction with Pointwise Circumscription
Koji Iwanuma, Katsumi Inoue and Hidetomo Nabeshima

15.15 Using Abduction and Induction for Operational Requirements Elaboration
Dalal Alrajeh, Oliver Ray, Alessandra Russo and Sebastian Uchitel

15.45 COFFEE

16.15 Equivalence Issues in Abduction and Induction
Chiaki Sakama and Katsumi Inoue

16.45 PANEL DISCUSSION

18.00 CLOSE

19.30 DINNER

Table of Contents

Using Abduction and Induction for Operational Requirements Elaboration

Dalal Alrajeh[1], Oliver Ray[1,2], Alessandra Russo[1], and Sebastian Uchitel[1,3]

[1] Imperial College London
{da04,or,ar3,su2}@doc.ic.ac.uk
[2] University of Bristol
oray@cs.bris.ac.uk
[3] University of Buenos Aires/CONICET
s.uchitel@dc.uba.ar

Abstract. Requirements Engineering involves the elicitation of high-level stakeholder goals and their refinement into operational system requirements. A key difficulty is that stakeholders typically convey their goals indirectly through intuitive narrative-style scenarios of desirable and undesirable system behaviour, whereas goal refinement methods usually require goals to be expressed declaratively using, for instance, a temporal logic. In actual software engineering practice, the extraction of formal requirements from scenario-based descriptions is a tedious and error-prone process that would benefit from automated tool support. This paper presents an Inductive Logic Programming (ILP) method for inferring operational requirements from a set of example scenarios and an initial but incomplete requirements specification. The approach is based on translating the specification and the scenarios into an event-based logic programming formalism and using a non-monotonic reasoning system, called eXtended Hybrid Abductive Inductive Learning (XHAIL), to automatically infer a set of event pre-conditions and trigger-conditions that cover all desirable scenarios and reject all undesirable ones. This learning task is a novel application of logic programming to requirements engineering that also demonstrates the utility of non-monotonic learning and the use of integrity constraints during generalisation for capturing the inter-relationship between pre-conditions and trigger-conditions.

1 Introduction

Requirements Engineering (RE) is an integral part of the software engineering life-cycle. It is concerned with the elicitation, elaboration, specification, analysis and documentation of goals concerning an envisaged system. These concerns all play important roles in the development of a complete and correct system specification. A core aspect of RE is the operationalisation of high-level goals. This process involves extracting from partial system descriptions (e.g. scenarios, goals, use cases, etc.,) operational requirements that satisfy high-level system goals as specified by the stakeholders. Few approaches have been developed to support the elicitation and elaboration of such requirements. These approaches focus

on the process of refining high-level goals into operational requirements [11, 12] declaratively expressed in a temporal logic [15]. The use of a temporal formalism enables the deployment of automated analysis and refinement tools, but is not directly accessible to most stakeholders with a less technical background. In fact, stakeholders prefer to convey their goals through more intuitive narrative-style scenarios of desirable and undesirable system behaviour [31] rather than temporal assertions. Because scenarios are inherently *partial* descriptions about specific system behaviours, they leave requirements implicitly defined. It is therefore necessary to synthesise declarative specifications of operational requirements that admit the desired behaviours while rejecting the undesired ones. Currently, the elicitation of declarative requirements from scenario-based descriptions is a tedious and error-prone process that relies on the manual efforts of an experienced engineer and would benefit from automated tool support.

This paper addresses this problem by providing a formal Inductive Logic Programming (ILP) [21] approach for *extracting* operational requirements from example scenarios and partial specifications. The approach here presented is an extension of the one introduced in [1], which used a non-monotonic ILP system to learn a particular type of operational requirements called event *pre-conditions* [12] from example scenarios and partial system description. In this paper, we show how this technique can, at the same time, be used to learn another class of operational requirements called event *trigger-conditions* [12]. Intuitively, pre-conditions state that a certain event *should not* happen under some conditions, while *trigger-conditions* are conditions under which an event *must* happen.

As in [1], the scenarios from which operational requirements are to be extracted represent examples of desirable and undesirable system behaviour over time; while a partial system specification captures an initial but incomplete background knowledge of the envisioned system and its environment. The task is to complete the specification by learning a set of missing event pre-conditions and trigger-conditions that covers all of the desirable scenarios, but none of the undesirable ones. The partial specification and scenarios are both expressed in Linear Temporal Logic (LTL) [15]. The former is expected to include information about the initial state of the system, domain properties and any pre-existing operational requirements, whereas the latter are sequences of events that can (or cannot) happen. Both, the partial specification and example scenarios, are transformed into an ILP representation based on the Event Calculus (EC) [7, 16] framework by means of a sound translation process. Because this representation makes essential use of negation in formalising the effects and non-effects of actions, the resulting learning problem is inherently non-monotonic.

We show that, under the stable model [5] semantics for logic programs with negation, the stable models of the transformed program correspond to the temporal models of the original specification. We then use the non-monotonic learning system XHAIL [25, 26] to generalise the scenarios with respect to the initial requirements specification. The abuctive component of XHAIL generates a ground unit explanation of the example scenaria, This preliminary explanation is then generalised by XHAIL's inductive component to produce the final hypothesis.

In this way, XHAIL integrates assumption-based and generalisaton-based inference within a coherent non-monotonic learning framework. The language bias of XHAIL is defined so as to allow the inference of pre- and trigger-conditions. The process of generalising operational requirements from scenarios makes use of integrity constraints for restricting the hypothesis space to include only those solutions that capture the inter-relationship between pre-conditions and trigger-conditions. We show that once the EC clauses learnt by XHAIL are translated back to LTL formulae, they provide a *correct extension* of the given partial system specifications with respect to the given example scenarios.

The paper is organized as follows. Section 2 presents the relevant background material on Inductive Logic Programming (ILP), XHAIL, Linear Temporal Logic (LTL), and the Event Calculus (EC). Section 3 describes the main features of our approach and presents some results on the soundness of the translation process of LTL requirements specifications into EC logic programs and on the correspondence between LTL and EC extensions of the initial specification. Section 4 provides an illustrative case study involving a Mine Pump controller, where event pre-conditions and trigger-conditions are simultaneously learned using the XHAIL system. A summary and some remarks about related and future work conclude the paper.

2 Background

This section introduces the necessary background material. After a summary of notation and terminology, two specific logic-based formalisms are described, namely Linear Temporal Logic (LTL), for expressing requirements specifications, and the Event Calculus (EC) for reasoning logically about action and change.

2.1 Notation and terminology

A term is either a constant, a variable or a function $F(t_1, ..., t_n)$ where F is a function symbol and t_i is a term. A *literal* is an atomic formula $\wp(t_1, ..., t_n)$ or its negation, *not* $\wp(t_1, ..., t_n)$ where $\wp$ is a predicate symbol, t_i is a term and *not* is the negation as failure operator. A *clause* is an expression of the form $\phi \leftarrow \psi_1 \wedge ... \wedge \psi_n$ where ϕ is an atom (called the head atom) and ψ_i is a literal (called a body literal). A clause is *ground* if it contains no variables. A clause is *definite* if all of its body literals are positive. The *empty clause* is denoted $\square$ and represents the truth value *false*. A *goal clause* is a clause $(\leftarrow \psi_1 ... \psi_n)$ with an empty head. A *logic program* is set of clauses. A definite logic program is a program in which all clauses are definite. A normal logic program is one in which the clauses are of the form $A \leftarrow B_1, \ldots, B_n$, *not* $C_1, \ldots,$ *not* C_m where A is the *head atom*, B_i are *positive body literals*, and *not* C_j are *negative body literals*.

In general, a *model* I of a normal logic program, Π, is a set of ground atoms such that, for each ground instance G of a clause in Π, I satisfies the head of G whenever it satisfies the body. A model I is *minimal* if it does not strictly include any other model. Definite programs always have a unique minimal model. Normal programs may have instead one, none, or several minimal models. It is

usual to identify a certain subset of these models, called *stable models*, as the possible meanings of the program. Given a normal logic program Π, the reduct of Π with respect to I, denoted Π^I, is the program obtained from the ground instances of Π by (a) removing all clauses with a negative literal *not* a in its body where $a \in I$ and (b) removing all negative literals from the bodies of the remaining clauses. Clearly Π^I is a definite logic program and as such has a unique minimal (Herbrand) model. If the model of Π^I coincides with I then I is said to be a stable model of Π as formalised in Definition 1.

Definition 1. *A model I of Π is a* stable model *if I is a minimal (Herbrand) model of Π^I where Π^I is the definite program $\Pi^I = \{A \leftarrow B_1, \ldots, B_n \mid A \leftarrow B_1, \ldots, B_n,\ not\ C_1, \ldots,\ not\ C_n$ is the ground instance of a clause in Π and I does not satisfy any of the $C_i\}$.*

2.2 Inductive Logic Programming and XHAIL

ILP is concerned with the computation of hypotheses H that generalise a set of (positive and negative) examples E with respect to a prior background theory B. In this paper we consider the case when B and H are normal logic programs, E is a set of ground literals (with positive and negative literals representing positive and negative examples, respectively), and H satisfies the condition $B \cup H \models E$ under the stable model semantics. In other words, the examples E must be satisfied in a stable model of $B \cup H$.[4] As formalised in Definition 2 below, it is usual to further restrict the clauses in H to a set of clauses S called a *hypothesis space*.

Definition 2. *Given a normal logic program B, a set of ground literals E, and a set clauses S, the task of ILP is to find a normal logic program $H \subseteq S$, consistent with B such that $B \cup H \models E$. In this case, H is called an* inductive generalisation *of E wrt. B and S.*

There are comparatively few ILP systems which can solve the learning task studied in this paper. In our experiments, we used the XHAIL framework [26], which is based on a three-phase Hybrid Abductive Inductive Learning (HAIL) approach [25]. XHAIL operates by constructing and generalising a preliminary ground hypothesis K, called a *Kernel Set* of B and E, which can be regarded as a non-monotonic multi-clause generalisation of the *Bottom Set* concept used in several well-known monotonic ILP systems [19, 20]. Like these monotonic ILP systems, XHAIL heavily exploits language and search bias when constructing and generalising a Kernel Set in order to bound the ILP hypothesis space.

The XHAIL language and search bias mechanisms are based upon the tried-and-tested notions of compression and mode declarations as used, for example, in Progol [19]. The compression heuristic favours hypotheses containing the fewest number of literals and is motivated by the scientific principle of Occam's razor

[4] Note that this condition assumes the consistency of B and H since at least one stable model of $B \cup H$ must exist in order for E to be satisfied.

(which, roughly speaking, means choose the simplest hypothesis that fits the data). Mode declarations provide a convenient mechanism for specifying which predicates may appear in the heads and bodies of hypothesis clauses and for controlling the placement and linking of constants and variables within those clauses [19].

As defined in [19], a mode declaration is either a head declaration $modeh(r, l)$ or a body declaration $modeb(r, s)$ where r is an integer (the recall) and s is a ground literal (the scheme) possibly containing so-called placemarker terms of the form $+t$, $-t$ and $\#t$, which must be replaced by input variables, output variables, and constants of type t, respectively. The symbol $*$ is often used to denote an arbitrary recall.

As explained in [26], XHAIL computes hypotheses using a nonmonotonic abductive interpreter to implement the three stages of the HAIL approach.[5] In the first phase, the head declarations are used to abduce the head atoms of the Kernel Set.[6] In the second phase, the body atoms of the Kernel Set are computed as the successful instances of queries obtained from the body declaration schemas. In the third phase, the hypothesis is computed by searching for a compressive theory that subsumes the Kernel Set, is consistent with the background knowledge, covers the examples and falls within the hypothesis space.[7]

2.3 Linear Temporal Logic

Several logic-based formalisms have been proposed for modeling event-based systems [6, 9, 27]. Among these Linear Temporal Logic (LTL) [15] is widely used and is supported by analysis techniques and tools such as model checking [14]. The language of LTL includes a finite non-empty set of Boolean propositions P, Boolean connectives $\neg, \wedge$ and $\rightarrow$ and temporal operators $\bigcirc$ (next), $\Box$ (always), and U (strong until). A well-formed LTL formula is constructed such that:

- *ff* and *tt*, representing truth and falsity respectively, are formulae.
- an atomic proposition p is a formula.
- if ϕ and ψ are formulae then so are: $\neg\phi$, $\phi \wedge \psi$, $\phi \rightarrow \psi$, $\bigcirc\phi$, $\Box\phi$, and $\phi \mathsf{U} \psi$.

Other formulae are introduced as abbreviations: $\phi \vee \psi$ abbreviates $\neg(\neg\phi \wedge \neg\psi)$ and $\phi \Leftrightarrow \psi$ abbreviates $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$. The formula $\Diamond\phi$ abbreviates $\neg\Box\neg\phi$, and $\phi W \psi$ abbreviates $(\Box\phi) \vee (\phi \mathsf{U} \psi)$. We use $(\neg)p$ to refer to either the proposition p or the negation $\neg p$ of that proposition. Also, we use $\bigcirc^i$ to denote i consecutive applications of the $\bigcirc$ operator. We further assume P to be partitioned into two sets P_e and P_f denoting *event* and *fluent* propositions respectively. The semantics of an LTL formula can be defined with respect to a structures called a Labeled Transition System (LTS) [30].

[5] There is a close correspondence between abduction and NAF [4].

[6] As an incremental approach is incompatible with the nonmonotonicity of normal programs, all examples (positive and negative) are processed by XHAIL in one go.

[7] While several pruning techniques can be used to prune the search space in the Horn case, these are unsound when applied to normal logic programs.

Definition 3 (Labeled Transition System). *A labeled transition system T is a tuple $(S, L, s_0, \rightarrow)$ where S is a finite nonempty set of states, L is a finite nonempty set of labels, called the* alphabet*, s_0 is a subset of S, called the* set of initial states*, and $\rightarrow \subseteq S \times L \times S$ is a nonempty set of transition relations.*

An *input* σ^* for an LTS T is a finite sequence of the form $l_1, l_2, ...l_m$ where $l_i \in L$. A *path* σ in T is a (possibly infinite) sequence $s_0, s_1, ...$, of states such that for each $i \geq 0$ there is a transition relation $(s_i, l_{i+1}, s_{i+1}) \in \rightarrow$, with label l_{i+1}. An input $\sigma^* = l_1, l_2, ...l_m$ is said to be *accepted* by T if there is a path $\sigma = s_0, s_1, ..., s_m$ in T where $(s_i, l_{i+1}, s_{i+1}) \in \rightarrow$ for all $0 \leq i < m$. Note that, for a given path (resp. input), the index of a state (resp. event) is referred to as its *position* in that path (resp. input).
An LTL model is a pair $\langle T, V \rangle$ consisting of an LTS, T, and a valuation function, V, that assigns to each fluent proposition an arbitrary set of pairs of path, in T, and positions in the path. The events, however, are not specified in V as their truth is implicitly determined by the transitions.

Definition 4. *Given an LTL language with propositions $P = P_e \cup P_f$, an LTL* model *is a pair $\langle T,\ V \rangle$ where T is an LTS with events P_e and V is a valuation function $V : P_f \Rightarrow 2^A$, where $A = \{(\sigma, i) \mid \sigma$ path in T and i position in $\sigma\}$.*

The satisfiability of an LTL formula in a model $M = \langle T, V \rangle$ is defined with respect to positions in a given path σ.

Definition 5. *Given an LTL language with propositions $P = P_e \cup P_f$, an LTL model $\langle T, V \rangle$ and a path σ in T, the satisfaction of an LTL formula ϕ at a position $i \geq 0$ of the path σ is defined inductively as follows:*

- $\sigma, 0 \not\models e$ *for any event proposition* $e \in P_e$
- $\sigma, i \models e$ *iff* e *is at position* i *(*$i \geq 1$*) in the path* σ*, where* $e \in P_e$
- $\sigma, i \models f$ *iff* $(\sigma, i) \in V(f)$*, where* $f \in P_f$
- $\sigma, i \models \neg\phi$ *iff* $\sigma, i \not\models \phi$
- $\sigma, i \models \phi \wedge \psi$ *iff* $\sigma, i \models \phi$ *and* $\sigma, i \models \psi$
- $\sigma, i \models \bigcirc\phi$ *iff* $\sigma, i+1 \models \phi$
- $\sigma, i \models \Box\phi$ *iff* $\forall j \geq i.\ \sigma, j \models \phi$
- $\sigma, i \models \phi\ \mathcal{U}\ \psi$ *iff* $\exists j \geq i.\ \sigma, j \models \psi$ *and* $\forall i \leq k < j.\ \sigma, k \models \phi$

An LTL formula ϕ is said to be *satisfied in a path* σ iff it is satisfied at the initial position, i.e. $\sigma, 0 \models \phi$. Similarly, a set of formulae Γ (also called a theory) is said to be satisfied in a path σ if each formula $\psi \in \Gamma$ is satisfied in the path σ. Given the above notions of satisfiability, we can now give a formal definition of a model of an LTL theory.

Definition 6 (Model of a Theory). *Let $M = (T, V)$ be an LTL model and Γ be an LTL theory. M is said to be a* model *of Γ, denoted $\models_M \Gamma$, iff Γ is satisfied in every path σ in T.*

Definition 7 (Entailment). *Let Γ be a LTL theory, ϕ a LTL formula and $M = \langle T, V \rangle$ an LTL model. The formula ϕ is said to be* entailed *by Γ, written $\Gamma \models_M \phi$, iff ϕ is satisfied in each path σ in T that satisfies Γ.*

2.4 Event Calculus

The Event Calculus (EC) formalism [7] is particularly well suited for reasoning about events and their effects over time [29] using logic programming techniques such as ILP. Its ontology is close enough to existing types of event-based requirements specifications to allow them to be mapped automatically into logical representations that can be used as a back-end to existing requirements engineering representational methods. in particular, an EC language includes three sorts of terms: *event* terms, *fluent* terms, and *time* terms. While time is represented by the non-negative integers $0, 1, 2, \ldots$, the event and fluent sorts are defined according to the domain being modeled. In this paper, we assume the EC language to include an additional sort called *scenarios*. The EC ontology includes the basic predicates *happens*, *initiates*, *terminates* and *holdsAt*. The atomic formula $happens(e, t, s)$ indicates that event e occurs at time-point t in a given scenario s, while $initiates(e, f, t, s)$ (resp. $terminates(e, f, t, s)$) means that, in a given scenario s, if event e were to occur at time t, it would cause fluent f to be true (resp. false) immediately afterwards. The predicate $holdsAt(f, t, s)$ indicates that fluent f is true at time-point t in a given scenario s. The formalism includes also an auxiliary predicate, *clipped*, and three additional predicates *impossible*, *attempt* and *triggered*. The predicate $clipped(t_1, f, t_2, s)$ means that, in a given scenario s, an event occurs which terminates f between times t_1 and t_2. The predicate $impossible(e, t, s)$ means that in a scenario s the event e cannot be performed at time t, the predicate $attempt(e, t, s)$ states that in a scenario s an event e may be attempted (i.e. is allowed) at time t, and $triggered(e, t, s)$ indicates instead that in a scenario s the event e has been triggered at time point t. Whereas the predicate *attempt* expresses the concept of a possible transition, the predicate *triggered* (resp. *impossible*) defines transitions that must (resp. cannot) occur. Their distinction is clearly captured by the translation function given in Section 3 for constructing EC programs from LTL requirement specifications.

An EC program includes domain dependent axioms describing which actions initiate and terminate which fluents, through the predicates *initiates* and *terminates*, which fluents are *initially* true, which events are *attempted*, and rules defining the *impossibility* and the *triggering* conditions of event occurrences. It also contains a narrative, which describes a course of events that may be attempted at specific time points and in specific scenarios, and a set of domain independent axioms which govern the interactions between the EC predicates:

$$clipped(T_1, F, T_2, S) \leftarrow happens(E, T, S), terminates(E, F, T, S), T_1 < T < T_2. \quad (1)$$

$$holdsAt(F, T_2, S) \leftarrow happens(E, T_1, S), initiates(E, F, T_1, S), T_1 < T_2, not\ clipped(T_1, F, T_2, S). \quad (2)$$

$$holdsAt(F, T, S) \leftarrow initially(F, S), not\ clipped(0, F, T, S). \quad (3)$$

$$happens(E, T, S) \leftarrow attempt(E, T, S), not\ impossible(E, T, S). \quad (4)$$

$$happens(E, T, S) \leftarrow attempt(E, T, S), triggered(E, T, S). \quad (5)$$

$$\leftarrow impossible(E, T, S), triggered(E, T, S). \quad (6)$$

The three axioms (1)-(3) describe general principles for deciding when fluents hold or do not hold at particular time-points[8]. They formalize the commonsense law of inertia which states that, a fluent that is true remains to hold until a terminating event occurs and vice versa. The two axioms (4) and (5) capture the semantics of event pre-conditions and event trigger-conditions respectively. The rule (4) states that an event E *cannot* happen if its pre-conditions are not satisfied (i.e. *impossible* is true). The rule (5), on the other hand, declares that an event E *must* happen if its trigger-conditions have been satisfied. The last rule (6) captures the semantic relationship between trigger-conditions and pre-conditions stating in particular that an impossible event can not be triggered at any time point in any scenario. An EC program is therefore a *normal logic program* with semantics given by the standard *stable model* semantics [5].

3 The Approach

In this section we show how ILP can be used to extend a partial system specification using information from a given set of scenarios. The learning problem is formalized in terms of an LTL specification. The given specification and scenarios are transformed, by means of a sound translation, into a non-monotonic ILP problem using an EC formalization which is then used to find a set of requirements that together with the partial specification account for the example scenarios.

3.1 Problem Description

Our aim is to develop an approach for extending a incomplete system specification with two types of operational requirements, namely event *pre-conditions* and *trigger-conditions* using information provided by user-defined scenarios. It is assumed that system specifications and scenarios are expressed in LTL. Since LTL is used in this context to represent system behaviors and their effects on the system and environment, it is further assumed that each fluent $f \in P_f$ is associated with two disjoint sets I_f and T_f called the *f-Initiating* set and *f-Terminating* set of event propositions respectively. Moreover, only a special class of LTL models are considered in which the labeled transition system T has a single initial state and the valuation function V is defined to capture the dependencies between fluent propositions and event propositions. For convenience, the notation E_{I_f} is used to represent the disjunction $\bigvee_{e \in I_f} e$ of f-Initiating events, and E_{T_f} for the disjunction $\bigvee_{e \in T_f} e$ of f-Terminating events. S_0 is instead used to represent the set of fluents $f \in P_f$ that are true in the initial system state s_0.

[8] These axioms are identical to those presented in [29] apart from the extra argument S for representing scenarios.

A *system specification* describes the relationship between events and fluents of the system and environment. As formalised below in Definition 8, it contains *initial state axioms* (7)-(8), specifying the fluent propositions that are true (resp. false) in the initial state; *persistence axioms* (9)-(10), formalising the commonsense law of inertia that any fluent will remain true (resp. false) until a terminating (resp. initiating) event occurs that causes it to flip truth value; *effect axioms* (11)-(12), which describe the effect a set of f-Initiating and f-Terminating events has on a fluent f; and finally a set of *pre-condition axioms* (13), and a set of *trigger-condition axioms* (14), describing the conditions under which an event *cannot* occur and *must* occur, respectively.

Definition 8. *A* system specification *is an LTL theory consisting of*

- *a positive initial state axiom representing all fluent propositions that are true in S_0 (if any).*
$$\bigwedge_{f_i \in S_0} f_i \tag{7}$$
- *a negative initial state axiom representing all fluent propositions that are not true in s_0 (if any).*
$$\bigwedge_{f_j \in P_f - S_0} \neg f_j \tag{8}$$
- *a pair of persistence axioms for each fluent proposition $f \in P_f$.*
$$\Box(f \rightarrow f \ \mathcal{W} \ E_{T_f}) \tag{9}$$
$$\Box(\neg f \rightarrow \neg f \ \mathcal{W} \ E_{I_f}) \tag{10}$$
- *a pair of f-Initiating and f-Terminating effect axioms for each fluent proposition $f \in P_f$*
$$\Box(E_{I_f} \rightarrow f) \tag{11}$$
$$\Box(E_{T_f} \rightarrow \neg f) \tag{12}$$
- *pre-condition axioms (if any).*
$$\Box(\bigwedge_{0 \leq i \leq n} (\neg) f_i \rightarrow \bigcirc \neg e) \tag{13}$$
- *trigger-condition axioms (if any).*
$$\Box(\bigwedge_{0 \leq i \leq m} (\neg) f_i \rightarrow \bigcirc e) \tag{14}$$

A scenario is a finite sequence of events $\langle e_1, ..., e_n \rangle$ (i.e. an input) that describes a system's behavior from its initial state. For each scenario, an associated *scenario property* can be defined. This can be of two types, namely an *existential scenario property*, expected to hold for some paths in T of a given model M of a system specification, or a *universal scenario property*, expected to hold over all the paths in T of a given model M of a system specification. The syntactic definition of these properties is given in Definition 9 and Definition 10 respectively.

Definition 9. *An* existential scenario property *is a property of the form (15), below, that is satisfied in some (i.e., at least one) path of a model* $M = \langle T, V \rangle$

$$\bigwedge_{1 \leq i \leq m-1} \bigcirc^i e_i \wedge \bigcirc^m (\neg) e_m \tag{15}$$

We refer to $(\bigwedge_{1 \leq i \leq m-1} \bigcirc^i e_i)$ as the *prefix* of the existential scenario property and to the sub-formula $\bigcirc^m (\neg) e_m$ as its *consequence*.

Definition 10. *A* universal scenario property *is a property of the form (16), below, that is satisfied in all paths of a model* $\langle T, V \rangle$

$$\bigwedge_{1 \leq i \leq n-1} \bigcirc^i e_i \rightarrow \bigcirc^n (\neg) e_n \tag{16}$$

The antecedent of a universal scenario property is also referred to as the *prefix* of the property. Definition (11) below formalizes the notion of *consistent* (existential and universal) scenario properties.

Definition 11. *Let* SP^u *and* SP^e *be sets of universal and existential scenario properties, respectively. The set* $SP^u \cup SP^e$ *is said to be a* consistent set *of scenario properties iff:*

- *There is no pair of universal scenario properties* $sp^u_1, sp^u_2 \in SP^u$ *with same prefixes and complementary consequences.*
- *There is no pair of universal scenario properties* $sp^u_1, sp^u_2 \in SP^u$ *with same prefixes and different positive consequences.*
- *For each universal scenario property* $sp^u \in SP^u$ *there is no existential scenario property* $sp^e \in SP^e$ *with the same prefix but complementary consequence.*
- *For each universal scenario property* $sp^u \in SP^u$ *there is no existential scenario property* $sp^e \in SP^e$ *with same prefix but different positive consequences.*

Given the above formalization, we can now define the task of learning preconditions and trigger-conditions.

Definition 12. *Let Spec be a system specification,* $M = \langle T, V \rangle$ *be a model of* Spec*,* SP^u *be a set of universal scenario properties of the form* $(\bigwedge_{1 \leq i \leq n-1} \bigcirc^i e_i \rightarrow \bigcirc^n \neg e_n)$*, and* SP^e *be a set of existential scenario properties of the form* $(\bigwedge_{1 \leq i \leq m} \bigcirc^i e_i)$ *such that* $(SP^u \cup SP^e)$ *is consistent. A set* Pre *of pre-condition axioms is a* correct pre-condition extension *of* Spec *with respect to* SP^u *and* SP^e *iff*

- $Spec \cup Pre \models_M sp^u$, *for each universal scenario property* $sp^u \in SP^u$,
- $Spec \cup Pre \not\models_M \neg sp^e$, *for each existential scenario property* $sp^e \in SP^e$.

Definition 13. *Let Spec be a system specification,* $M = \langle T, V \rangle$ *be a model of* Spec*,* SP^u *be a set of universal scenario properties of the form* $(\bigwedge_{1 \leq i \leq n-1} \bigcirc^i e_i \rightarrow \bigcirc^n e_n)$ *and* SP^e *be a set of existential scenario properties of the form* $(\bigwedge_{1 \leq i \leq m-1} \bigcirc^i e_i \wedge \bigcirc^m \neg e_m)$ *such that* $(SP^u \cup SP^e)$ *is consistent. A set* Trig *of trigger-condition axioms is a* correct trigger extension *of Spec with respect to* SP^u *and* SP^e *iff*

- $Spec \cup Trig \models_M sp^u$, *for each universal scenario property* $sp^u \in SP^u$,
- $Spec \cup Trig \not\models_M \neg sp^e$, *for each existential scenario property* $sp^e \in SP^e$.

Now we define the task for learning a correct extension.

Definition 14. *Let Spec be a system specification,* $M = \langle T, V \rangle$ *be a model of* Spec, $SP^u \cup SP^e$ *be a set of consistent scenario properties. A set* $Pre \cup Trig$ *of pre-condition and trigger-condition axioms is a* correct extension *of Spec with respect to* SP^u *and* SP^e *iff*

- $Spec \cup Pre \cup Trig \models_M sp^u$, *for each* $sp^u \in SP^u$,
- $Spec \cup Pre \cup Trig \not\models_M \neg sp^e$, *for each* $sp^e \in SP^e$.

Intuitively, finding a correct extension of a given specification with respect to given existential and universal scenario properties means *refining* the models of the original *Spec* by removing unwanted or undesirable traces. Given a model $M = (T, V)$ of a specification *Spec*, adding a pre-condition axiom to *Spec* has the effect of removing from T all those (sub-)paths that satisfy at the first position the prefix of the pre-condition and have a first transition labeled with the event in the consequence of axiom. On the other hand, adding a trigger-condition axiom to the *Spec* means removing from T all those (sub-)paths that satisfy at the first position the prefix of the trigger-condition and have a first transition labeled with an event different from the event in the consequence of axiom.

3.2 Translating LTL Specifications into EC Logic Programs

In order to apply ILP to the task of learning correct extensions, the LTL system specification and scenario properties, of the form defined above, are translated into EC normal logic program, where the sorts of event and fluent are given by the set P_e of event propositions and the set P_f of fluent proposition of the LTL language; time points corresponding to the positions in the paths of a model M, and a scenario constant is introduced for every universal and existential scenario property in SP^u and SP^e. The translation of a system specification *Spec* into such a corresponding EC program Π is formally defined below.

Definition 15. *Given a system specification Spec expressed in LTL, the* corresponding *logic program* $\Pi = \tau(Spec)$ *is defined as the program containing the following clauses:*

- *the fact* initially(f_i,S) *for each fluent* f_i *appearing in a positive initial state axiom of the form* $\bigwedge_{f_i \in S_0} f_i$

- *the fact* initiates(e_i,f,T,S) *for each* f*-Initiating event* $e_i \in I_f$ *appearing in an* f*-Initiating effect axiom of the form* $\Box(E_{I_f} \rightarrow f)$

- *the fact* terminates(e_i,f,T,S) *for each* f*-Terminating event* $e_i \in T_f$ *appearing in a* f*-Terminating effect axiom of the form* $\Box(E_{T_f} \rightarrow \neg f)$

– *the clause* impossible(e,T,S)← $\bigwedge_{0\leq i\leq k}(not)$holdsAt($f_i$,T,S) *for each pre-condition axiom of the form* $\Box(\bigwedge_{0\leq i\leq k}(\neg)f_i \rightarrow \bigcirc\neg e)$

– *the clause* triggered(e,T,S)← $\bigwedge_{0\leq i\leq l}(not)$holdsAt($f_i$,T,S) *for each trigger-con-dition axiom of the form* $\Box(\bigwedge_{0\leq i\leq l}(\neg)f_i \rightarrow \bigcirc e)$

– *the EC core axioms (1)-(6).*

Note that negative initial state axioms (8) and persistence axioms (9) and (10) of an LTL specification are all implicitly captured by the stable model interpretation of the EC core axioms. Moreover, the semantics of the temporal operator $\Box$ is captured by the implicit universal quantification over the time variable T that appears in the *initiates* and *terminates* facts and in the *impossible* and *triggered* rules. Theorem 1 below states that the translation τ in Definition 15 is sound, i.e. for any path σ in a given model M of *Spec* there is a corresponding narrative Nar such that the program $\Pi = \tau(Spec) \cup Nar$ satisfies the same fluent and event formulae as in σ.

Theorem 1 (Soundness and Completeness). *Let Spec be a system specification and $M = \langle T, V\rangle$ be a model of Spec. Let $\sigma = \langle e_1, e_2, \ldots, e_n\rangle$ be an accepted input in T and let Nar be the set of facts of the form attempt$(e_i, i-1, \sigma)$ for each event e_i in σ. Let Π be the EC logic program $\Pi = \tau(Spec) \cup Nar$ with a unique stable model I. Then, for any fluent f and position i, we have $\sigma, i \models f$ iff holdsAt(f, i, σ) is true in I; and, for any event e and position i, we have $\sigma, i \models e$ iff happens$(e, i-1, \sigma)$ is true in I.*

Proof. The proof is by induction of the position i in σ using the fact that Π is a locally stratified program and, as such, has a unique stable model [3].

3.3 Learning Requirements using ILP

ILP is concerned with the task of learning a hypothesis H that that explains a set of examples E with respect to a background theory B. In the context of learning requirements, we are given a partial specification B a set of scenarios E, the task is to learn a set H of operational requirements such that $B \cup H \models E$ (under the stable model semantics). In the EC formalism, event pre-conditions are represented as clauses with *impossible* in the head and *holdsAt* literals in the body

$$impossible(E, T, S) \leftarrow \bigwedge_{0\leq i\leq n}(not)\ holdsAt(f_i, T, S) \tag{17}$$

while event trigger-conditions are represented as clauses of the form

$$triggered(e, T, S) \leftarrow \bigwedge_{0\leq i\leq m}(not)\ holdsAt(f_i, T, S) \tag{18}$$

Hence, the task of learning requirements is the process of generating hypotheses of the form above from a partial specification and a set of scenario properties which, respectively, comprise the background and examples. The function

τ can be used to translate an initial LTL specification into an ILP theory. To fully define the inductive learning task a corresponding translation from scenario properties to ILP example must be specified. As shown in Definition 16 below, scenario properties contribute to the background theory as well as to the examples. The translation depends on the event for which the pre-condition (resp. trigger-condition) axiom is to be learnt. In what follows, it is assumed that pre-conditions (resp. trigger-conditions) axioms are to be learnt for the last event of each universal scenario and existential scenario property. Therefore, in the case of pre-conditions (resp. trigger-conditions), each universal scenario property produces a sequence of facts stating that certain events do happen followed by one fact stating that some particular event should not (resp. must) happen immediately afterward and each existential scenario property simply states that a certain sequence of events does (resp. does not) happen.

Definition 16. *Given a system specification* Spec *and a set of consistent universal and existential scenario properties $SP^u \cup SP^e$, the* EC translation $\tau(Spec, SP^u, SP^e)$ *is the pair (B, E) of EC programs constructed as follows:*

- *for each universal scenario property $sp^u = (\bigcirc e_1, ... \wedge \bigcirc^{n-1} e_{n-1} \rightarrow \neg \bigcirc^n e_n)$ in SP^u*
  - *E includes n-1 facts of the form $happens(e_i, i-1, sp^u)$ with $1 \leq i < n$.*
  - *E includes 1 fact of the form not $happens(e_n, n-1, sp^u)$.*
  - *B includes n facts $attempt(e_i, i-1, sp^e)$ with $1 \leq i \leq n$.*
- *for each universal scenario property $sp^u = (\bigcirc e_1 \wedge ... \wedge \bigcirc^{m-1} e_{m-1} \rightarrow \bigcirc^m e_m)$ in SP^u:*
  - *E includes m facts of the form $happens(e_i, i-1, sp^u)$ with $1 \leq i \leq m$.*
  - *B includes m facts of the form $attempt(e_i, i-1, sp^u)$ with $1 \leq i \leq m$.*
- *for each existential scenario property $sp^e = \bigwedge_{1 \leq i \leq l-1} \bigcirc^i e_i \wedge \bigcirc^l \neg e_l$ in SP^e*
  - *E includes l-1 facts of the form $happens(e_i, i-1, sp^e)$ with $1 \leq i < l$.*
  - *E includes 1 fact of the form not $happens(e_l, l-1, sp^e)$.*
  - *B includes l facts of the form $attempt(e_i, i-1, sp^e)$ with $1 \leq i \leq l$.*
- *for each existential scenario property $sp^e = \bigwedge_{1 \leq i \leq k} \bigcirc^i e_i$ in SP^e*
  - *E includes k facts of the form $happens(e_i, i-1, sp^e)$ with $1 \leq i \leq k$.*
  - *B includes k facts of the form $attempt(e_i, i-1, sp^e)$ with $1 \leq i \leq k$.*
- *B includes all facts and rules in $\tau(Spec)$.*

As shown in Definition (16), given a partial specification *Spec* and sets SP^u and SP^e of universal and existential scenario properties respectively, the translation τ results in corresponding EC programs consisting of a background theory B, and a set of examples E where $(B, E) = \tau(Spec, SP^u, SP^e)$.

The utility of the transformation is demonstrated by Theorem 2, which shows that τ can be used to compute correct extensions via nonmonotonic ILP. Given a partial specification *Spec*, and scenario properties SP^u and SP^e, the corresponding EC programs B and E are obtained using τ and the hypothesis space S is defined as the set of clauses of the form (17) and (18). Theorem 2 states that any ILP solution to this problem can be translated back into a correct LTL extension of the initial *Spec* with respect SP^e and SP^u.

Theorem 2. *Let Spec be a system specification, let $SP^u \cup SP^e$ be a set of consistent scenario properties, let $(B, E) = \tau(Spec, SP^u, SP^e)$ be their EC translation, and let S be the set of clauses of the form (17) and (18) — i.e. the set of all event pre-condition and trigger-condition rules. Then, for any inductive generalisation H of E wrt. B and S, the corresponding set $Pre \cup Trig = \tau^{-1}(H)$ of LTL pre-condition and trigger-condition axioms is a correct extension of Spec with respect to SP^u and SP^e.*

In other words, for a given B and E, any inductive solution of the form (17) and (18) once translated back into LTL and added to the original partial specification will have the effect of eliminating all paths in T which violates sp^u but still contains at least one path satisfying sp^e. Note that depending on the generality of H, $\tau^{-1}(H)$ may also eliminate paths in T which satisfy sp^u. This is restricted by the fact that $\tau^{-1}(H)$ should cover with all universal and existential scenarios considered. In order to extend a partial specification in this way, we use the nonmonotonic ILP system XHAIL [26], as illustrated in the following case study.

4 Case Study: A Mine Pump Control System

This section presents an extension of case study used in [1] as an application of the learning approach proposed in this paper to a event-driven system involving a Mine Pump Controller [8]. This is a system that is supposed to monitor and control water levels in a mine, to prevent water overflow. It is composed of a pump for pumping mine-water up to the surface as well sensors for monitoring the water levels and methane percentage. The pump should be activated once the water has reached pre-set high water level and deactivated once it reaches low water level. Moreover, the pump should be switched off if the percentage of methane in the mine exceeds a certain critical limit.

An initial partial system specification *Spec* is given along with a set of positive and negative scenario properties, written in an LTL language with fluent propositions $P_f = \{pumpOn, criticalMethane, highWater\}$ and event propositions $P_e = \{turnPumpOn, turnPumpOff, signalCriticalMethane, signalNotCriticalMethane, signalHighWater, signalNotHighWater\}$. The specification includes information about the initial state of the system, persistence axioms, effect axioms and a single trigger-condition axiom, all formalised as follows:

$$(\neg criticalMethane \wedge \neg pumpOn \wedge \neg highWater) \tag{19}$$

$$\Box(criticalMethane \rightarrow (criticalMethane \ \mathsf{W} \ signalNotCriticalMethane)) \tag{20}$$

$$\Box(\neg criticalMethane \rightarrow (\neg criticalMethane \ \mathsf{W} \ signalCriticalMethane\ )) \tag{21}$$

$$\Box(pumpOn \rightarrow \ (pumpOn \ \mathsf{W} \ turnPumpOff)) \tag{22}$$

$$\Box(\neg pumpOn \rightarrow \ (\neg pumpOn \ \mathsf{W} \ turnPumpOn)) \tag{23}$$

$$\Box(highWater \rightarrow (highWater \ \ \mathsf{W} \ signalNotHighWater)) \tag{24}$$

$$\Box(\neg highWater \rightarrow (\neg highWater \ \mathsf{W} \ signalHighWater)) \tag{25}$$

$$\Box(signalCriticalMethane \rightarrow criticalMethane) \tag{26}$$
$$\Box(signalNotCriticalMethane \rightarrow \neg criticalMethane) \tag{27}$$
$$\Box(signalHighWater \rightarrow highWater) \tag{28}$$
$$\Box(signalNotHighWater \rightarrow \neg highWater) \tag{29}$$
$$\Box(turnPumpOn \rightarrow pumpOn) \tag{30}$$
$$\Box(turnPumpOff \rightarrow \neg pumpOn) \tag{31}$$
$$\Box(methane \rightarrow \bigcirc turnPumpOff) \tag{32}$$

Equation (19) defines the negative initial state of the system, equations (20)–(25) specify the persistence axioms, equations (26)–(31) define the effect axioms and finally equation (32) specifies a trigger-condition axiom. Note that because all fluents are assumed to be initially false, the specification does not include a positive initial state axiom.

A consistent set of universal and existential scenario properties for this system is given by the following formulae:

$$sp_1^u = (\bigcirc signalCriticalMethane \wedge \bigcirc^2 signalHighWater \rightarrow \bigcirc^3 \neg turnPumpOn) \tag{33}$$
$$sp_1^e = (\bigcirc \neg turnPumpOn) \tag{34}$$
$$sp_2^e = (\bigcirc signalHighWater \wedge \bigcirc^2 turnPumpOn) \tag{35}$$

Applying the translation τ to the specification and scenario properties above results in an ILP theory B composed of the EC core axioms and the following clauses:

initiates(signalCriticalMethane,criticalMethane,T,S).
terminates(signalNotCriticalMethane,criticalMethane,T,S).
initiates(signalHighWater,highWater,T,S).
terminates(signalNotHighWater,highWater,T,S).
initiates(turnPumpOn,pumpOn,T,S).
terminates(turnPumpOff,pumpOn,T,S).
triggered(turnPumpOff,T,S) $\leftarrow$ *holdsAt(methane,T,S).*

attempt(signalCriticalMethane,0,sp_1^u*).* *attempt(signalHighWater,1,*sp_1^u*).*
attempt(turnPumpOn,2,sp_1^u*).* *attempt(turnPumpOn,0,*sp_1^e*).*
attempt(signalHighWater,0,sp_2^e*).* *attempt(turnPumpOn,1,*sp_2^e*).*

In addition, the translation produces the following set of ILP examples E:

happens(signalCriticalMethane,0,sp_1^u*).* *happens(signalHighWater,1,*sp_1^u*).*
not happens(turnPumpOn,2,sp_1^u*).* *not happens(turnPumpOn,0,*sp_1^e*).*
happens(signalHighWater,0,sp_2^e*).* *happens(turnPumpOn,1,*sp_2^e*).*

In order bias XHAIL to learn pre- and trigger-conditions, the following mode declarations are used.

$$\begin{array}{l} modeh(*,\ impossible(\#event, +time, +scenario)) \\ modeh(*,\ triggered(\#event, +time, +scenario)) \\ modeb(*,\ holdsAt(\#fluent, +time, +scenario)) \\ modeb(*,\ not\ holdsAt(\#fluent, +time,\ +scenario)) \end{array} \tag{36}$$

As explained in [19] these ensure the hypothesis space S consists of clauses of the form (17) and (18) with atoms $impossible(e, T, S)$ and $triggered(e, T, S)$ in the head and literals of the form (not) $holdsAt(e, T, S)$ in the body. Applying XHAIL to B and E then yields the (non minimal) abductive explanation $\Delta =$

$$\begin{array}{l} triggered(turnPumpOn, 1, sp_2^e) \\ impossible(turnPumpOn, 2, sp_1^u) \\ impossible(turnPumpOn, 0, sp_1^e) \end{array} \tag{37}$$

which is turned into the Kernel Set $K =$

$$\begin{array}{l} triggered(turnPumpOn, 1, sp_2^e) \leftarrow holdsAt(highWater, 1, sp_2^e), \\ \qquad not\, holdsAt(pumpOn, 1, sp_2^e), not\, holdsAt(methane, 1, sp_2^e). \\ impossible(turnPumpOn, 2, sp_1^u) \leftarrow holdsAt(highWater, 2, sp_1^u), \\ \qquad not\, holdsAt(pumpOn, 2, sp_1^u), holdsAt(methane, 2, sp_1^u). \\ impossible(turnPumpOn, 0, sp_1^e) \leftarrow not\, holdsAt(highWater, 0, sp_1^e), \\ \qquad not\, holdsAt(pumpOn, 0, sp_1^e), not\, holdsAt(methane, 0, sp_1^e). \end{array} \tag{38}$$

and is generalised to give the maximally compressive hypotheses $H =$

$$\begin{array}{l} triggered(turnPumpOn, X, Y) \leftarrow holdsAt(highWater, X, Y), \\ \qquad not\, holdsAt(methane, X, Y). \\ impossible(turnPumpOn, X, Y) \leftarrow holdsAt(methane, X, Y). \\ impossible(turnPumpOn, X, Y) \leftarrow not\ holdsAt(highWater, X, Y). \end{array} \tag{39}$$

corresponding to the correct extension

$$\begin{array}{l} \Box((highWater \wedge \neg methane) \rightarrow \bigcirc turnPumpOn) \\ \Box(methane \rightarrow \bigcirc \neg turnPumpOn) \\ \Box(\neg highWater \rightarrow \bigcirc \neg turnPumpOn) \end{array} \tag{40}$$

stating that the pump should turn on whenever there is water and no methane; and that it should turn off whenever there is methane or no water.

Note that the integrity constraint (6) plays an important role since leaving it out would result in the more compressive solution

$$\begin{array}{l} \Box((highWater \wedge \neg methane) \rightarrow \bigcirc turnPumpOn) \\ \Box(\bigcirc \neg turnPumpOn) \end{array} \tag{41}$$

in which $turnPumpOn$ would have to be both true and false whenever $highWater$ was true but $methane$ was false.

5 Related Work

Among the few approaches that use inductive learning for requirements elicitation it worth mentioning [10]. The approach presented in [10] captures high-level goals, as temporal formulae, from manually tailored scenarios provided by

stake-holders using an *ad-hoc* inductive inference process. The tailored scenarios are then used to elicit new declarative goals that explain the given scenario. These new goals are then added to the given initial (partial) goal model for "non-operational" analysis (i.e. goal decomposition, conflict management and obstacle detection). The inductive inference of these declarative goals is simply a process of pure generalization of the given scenarios that does not take into account the given partial goal model. It is therefore a potentially unsound inference process in that it can generate declarative goals that are inconsistent with the given (partial) goal model. Our approach, on the other hand, can be extended so to include the goals as integrity constraints. In so doing, the learnt *operational* requirements would be guaranteed to be consistent with the given goals and partial specification and can therefore be directly added to it.

The ILP task defined in this paper is somewhat related to some earlier work in [18] and [17], where the ILP systems Progol5 and Alecto were applied to the learning of domain specific EC axioms. Like XHAIL, these procedures employ an abductive reasoning module to enable the learning of predicates distinct from those in the examples — an ability that is clearly required in this application. However, unlike XHAIL, they do not have a well-defined semantics for non-definite programs and their handling of negation is rather limited [26]. In fact, the inability of Progol5 and Alecto to reason abductively through nested negations means that neither of these systems can solve the case study presented in this paper. Some related approaches for inferring action theories from examples are presented in [13], [22] and more recently in [24], which reduce learning in the Situation Calculus to a monotonic ILP framework. These approaches work by pre- and post-processing the inputs and outputs of a conventional Horn Clause ILP system. This technique is very efficient, but is not as general as our own approach. An alternative method for nonmonotonic ILP under the stable model semantics is proposed in [28], but cannot be used in our case study because it assumes the target predicate is the same as the examples. [28] also includes a thorough review of previous work on non-monotonic ILP. A more recent technique is proposed in [23] that uses a combination of SAT solvers and Horn ILP to perform induction under the stable model semantics.

6 Conclusion and Future Work

This paper presents a methodology for extending a partial system specification with event pre-conditions and trigger-conditions from information provided by user scenarios using ILP. This involves transforming the initial specification and scenarios from an LTL representation into an EC logic programs which are then used by a non-monotonic ILP system to learn the missing requirements. By exploiting the semantic relationship between the LTL and EC, the approach thereby provides a sound ILP computational "back-end" to a temporal formalism familiar to Requirements Engineers.

It is assumed in this paper that scenarios are provided by stake-holders. Current work involves integrating ILP and model checking techniques, as the ones presented in [2, 11], such that undesirable scenarios are generated automatically using model checking tools. A current assumption of the approach described in this paper is that the scenarios provided to the ILP technique are complete in the sense that the event appearing in the sequence are the only events that occur. An area for future research is to relax these assumptions and learn operational requirements from *incomplete* scenarios that satisfy a given specification. Furthermore, system specifications are assumed to be asynchronous. By that we mean that the specification is expected to be satisfied at every position in a path of an LTL model. In goal-oriented requirements engineering approaches, system specifications are usually represented synchronously in which the specification is assumed to hold at certain time points rather than positions. We therefore aim to extend the approach to handle learning synchronous specifications as well as asynchronous. Future research also includes extending the specification with other forms of operational requirements such as postconditions which capture additional conditions on fluents that must (not) hold as a consequence of executing event e written as $\Box(e \rightarrow \bigwedge_{1 \leq i \leq n}(\neg)f)$. In addition, a main focus of interest is to incorporate user-defined goals in the learning process to guarantee that the learnt requirements satisfy the stake-holders goals.

Acknowledgments This work is funded by the Philip Leverhulme Trust, Research Councils UK and the Saudi Arabian Ministry of Higher Education.

References

1. D. Alrajeh, O. Ray, A. Russo, and S. Uchitel. Extracting requirements from scenarios with ILP. In *16th Int. Conference on Inductive Logic Programming*, pages 63–77, 2006.
2. D. Alrajeh, A. Russo, and S. Uchitel. Inferring operational requirements from goal models and scenarios using inductive systems. In *Proc. 5th Int. Workshop on Scenarios and State Machines*, 2006.
3. K. R. Apt and R.N. Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.
4. K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proc. of the 6th Int. Conf. on Logic Programming*, pages 234–254, 1989.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R.A. Kowalski and K. Bowen, editors, *Proc. of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. The MIT Press, 1988.
6. D. Giannakopoulou and J. Magee. Fluent model checking for event-based systems. In *Proc. 11th ACM SIGSOFT Symp. on Foundations Software Engineering*, 2003.
7. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New generation computing*, 4(1):67–95, 1986.
8. J. Kramer, J. Magee, and M. Sloman. Conic: An integrated approach to distributed computer control systems. In *IEE Proc., Part E 130*, pages 1–10, Jan. 1983.
9. A. Van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. 5th IEEE Int. Symp. on Requirements Engineering*, pages 249–263, 2001.

10. A. Van Lamsweerde and L. Willemet. Inferring declarative requirements specifications from operational scenarios. *IEEE Trans. on Software Engineering*, 24(12):1089–1114, 1998.
11. E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transitions systems from goal-oriented requirements models. Technical Report 2006/2, Imperial College London, 2005.
12. E. Letier and A. Van Lamsweerde. Deriving operational software specifications from system goals. In *Proc. 10th ACM SIGSOFT Symp. on Foundations of Software Engineering*, pages 119–128, 2002.
13. D. Lorenzo. *Learning non-monotonic Logic Programs to Reason about Actions and Change*. PhD thesis, University of Coruna, 2001.
14. J. Magee and J. Kramer. *Concurrency : State Models and Java Programs*. John Wiley and Sons, 1999.
15. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.
16. R. Miller and M. Shanahan. Some alternative formulation of event calculus. *Computer Science; Computational Logic; Logic programming and Beyond*, 2408, 2002.
17. S. Moyle. *An investigation into Theory Completion Techniques in ILP*. PhD thesis, University of Oxford, 2000.
18. S. Moyle and S. Muggleton. Learning programs in the event calculus. In *Proc. 7th Int. Workshop on ILP*, 1997.
19. S.H. Muggleton. Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
20. S.H. Muggleton and C.H. Bryant. Theory Completion Using Inverse Entailment . In *Proc. 10th Int. Conf. on ILP*, volume 1866 of *LNCS*, pages 130–146. 2000.
21. S.H. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
22. R. Otero. Embracing causality in inducing the effects of actions. In *Proc. 10th Conf. of the Spanish Assoc. for AI*, 2004.
23. R. Otero and J. Gonzalez. Iaction: a system for induction under non-horn programs with stable models. In *Proc. of the 16th Int. Conf. on ILP*, volume submitted of *Lecture Notes in Artificial Intelligence*, 2006.
24. R. Otero and M. Varela. Iaction: a system for learning action descriptions for planning. In *Proc. of the 16th Int. Conf. on ILP*, volume submitted of *Lecture Notes in Artificial Intelligence*, 2006.
25. O. Ray. *Hybrid Abductive-Inductive Learning* . PhD thesis, Imperial College London, 2005.
26. O. Ray. Using abduction for induction of normal logic programs. In *Proc. ECAI'06 Workshop on Abduction and Induction in AI and Scientific Modelling*, pages 28–31, 2006.
27. A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. An abductive approach for analysing event-based requirements specifications. In *Proc. 18th Int. Conf. on Logic Programming*, volume 2401 of *LNCS*, pages 22–37, 2002.
28. C. Sakama. Induction from answer sets in non-monotonic logic programs. *ACM Trans. on Computational Logic*, 6(2):203–231, 2005.
29. M.P. Shanahan. *Solving the Frame Problem*. MIT Press, 1997.
30. C. Stirling. Comparing Linear and Branching Time Temporal Logics. In *Temporal Logics in Specification*, volume 398 of *LNCS*, pages 1–20. Springer Verlag, 1987.
31. A. Sutcliffe, N. A. M. Maiden, S. Minocha, and D. Manuel. Supporting scenario-based requirements engineering. *IEEE Trans. on Software Engineering*, 24:1072–1088, 1998.

Towards Refinement of Abductive or Inductive Hypothesis through Propagation

Gauvain Bourgne[1], Amal El Fallah Seghrouchni[2], and Nicolas Maudet[1]

[1] LAMSADE, Univ. Paris IX-Dauphine
Paris 75775 Cedex 16 (France)
Email: {bourgne,maudet}@lamsade.dauphine.fr
[2] LIP6, Univ. Pierre and Marie Curie
104, Avenue du Prsident Kennedy - 75016 - Paris (France)
Email: Amal.Elfallah@lip6.fr

Abstract. In this paper we address the problem of *distributed* sources of information (agents) that observe locally the environment, and have to communicate in order to refine their hypothesis regarding the actual state of this environment. One manner to attack the problem would be to centralize all the collected observations and knowledge, and to centrally compute the resulting theory. In many situations however, this would not be possible to adopt this centralized approach (*e.g.* for practical reasons, or privacy concerns). In this paper, we assume that agents individually face abductive or inductive tasks in a globally coherent environment, and we show that general mechanisms can be designed that abstractly regard both cases as special instances of a problem of *hypothesis refinement through propagation*. Assuming that agents are equipped with some individual revision machinery, our concern will be to investigate how (under what conditions) convergence to a consistent state can be guaranteed at more global levels: (i) between two agents; (ii) in a clique of agents; and (iii) in general in a connected society of agents.

1 Introduction

In this paper we address the problem of *distributed* sources of information (agents) that observe locally the environment, and have to communicate in order to refine their hypothesis regarding the actual state of this environment. One manner to attack the problem would be to centralize all the collected observations and datas, and to centrally compute the resulting theory. This trivially solves the problem if we assume that the situation is globally consistent, but still necessitates to merge knowledge bases if agents have potentially conflicting opinons [8].

In many situations however, this would not be possible to adopt this centralized approach; either because no agent would be prepared to play the role of a central authority; because the number and spatial repartition of agents makes that approach not realistic in practice; or because privacy concern makes agents reluctant to communicate some of their informations.

This would be the case for instance if different labs could hold observations (medical datas) regarding a patient, and had to carefully communicate with each others in order to come up with a satisfying diagnostic, while keeping some information private. One further illustration would be agents representing different knowledge bases storing examples linking genes and function, and agents communicating in order to refine their knowledge of the gene-function rule.

In the context of this paper, we assume that the environment is globally consistent. As the examples above suggest, the task that agents face can be either abductive or inductive, and we precisely argue in this paper that general mechanisms can be designed that abstractly regards both cases as special instances of a problem of *hypothesis refinement through propagation*. Section 2 introduces the formal framework and notions that we shall use throughout the paper. Agents will be supposed to be equipped with some individual revision machinery, and our concern will be to investigate how (under what conditions) consistence at a more global level (being it two agents, or within the whole system) can be guaranteed. As must be clear from our introduction, we will not adopt here a broadcast method that would amount to widely propagate agents' knowledge, but favour more focused mechanisms that seek to optimize information propagation. Section 3 present these protocols, and discuss their properties. The paper eventually shows how the framework can be instantiated in the context of an abductive or inductive application (Section 4). Section 5 draws some connections with related works. Section 6 concludes and reports on preliminary experiments that have been published in companion papers.

2 Formal Model

2.1 Agent

We take a *system* populated by n agents $a_1, \ldots, a_n$. Each agent a_i has two different kind of knowledge:

- K_i is the *information set*, representing all the *certain*, *non-revisable*, knowledge of the agent. Among these certain knowledge, we further distinguish two sets :
  - $\mathcal{F}$, the set of *facts*, is the common ground of the agents. It represents *prior knowledge* that are shared by all agents.
  - O_i is its *observation set*. We assume a perfect senses and memory, hence these observations are certain and the set grows monotonically. This set represents the *collected knowledge* of each individual agent, it contains all certain knowledge that are acquired by the agents. We note $\mathcal{O}$ the set of possible observations in the system. Therefore, $O_i \subseteq \mathcal{O}$.
- A_i is the *belief base*, or *assumed hypotheses set*, representing all the *uncertain*, explicit knowledge of the agent. These beliefs are usually derived from non-monotonic inferences and are therefore *revisable*. We can constrain this set by specifying the set $\mathcal{H}$ of *possible hypotheses*. We will note h_i the conjunction or disjunction (according to the application) of all the elements of A_i, called the *working hypothesis*, or *favourite hypothesis* of the agent.

Using these explicit knowledge, we can now derive the *implicit knowledge* of the agent. The *belief set* B_i, representing all the knowledge the agent *believes* to be true, with ou without certainty, is defined as $B_i = Cn(K_i \cup A_i)$ (where, at this point, $Cn(T)$ just represents the set of all possible conclusions that can be soundly derived from T.) Among this belief set B_i, we will distinguish the *closed information set* $\mathcal{K}_i = Cn(K_i)$ representing all the certain knowledge, that the agent *knows* to be true.

2.2 Consistency

As beliefs are uncertain, there can be contradiction between them. We want to ensure that the working hypothesis h_i (and by extension the belief set B_i) has some property ensuring its internal coherence and adequacy with the information set K_i. We shall then define an abstract notion of *consistency relation* to capture this adequacy. Depending of the type of beliefs or hypothesis, this relation might take different forms such as logical coherence, when considering belief revision, coherence and completeness of an abductive or inductive hypothesis, when building explanation or generating rules, or even arc-consistency in distributed constraint satisfaction problems. This notion is used to abstract away from the specifics of different applications and define communication protocols as generic as possible. We will represent it as a complete binary relation $Cons_\alpha(h, K)$ between an hypothesis $h \sqsubseteq \mathcal{H}$[3] and an information set K, where α will enable different instanciation for applications. As $\mathcal{F}$ is common to all agents and does not vary, we will take it out of the consistency notation, and abuse the notation a little bit by simply denoting by $Cons_\alpha(h, O)$ the basic consistency relation.

The only requirement that we put on the consistency relation so far is that it is assumed to be *compositional*, meaning that:

$$Cons_\alpha(h, O) \text{ and } Cons_\alpha(h, O') \text{ iff } Cons_\alpha(h, O \cup O') \tag{1}$$

To understand the consequences of this assumption, it is useful to distinguish both directions of this equivalence relation. The 'if' direction is often called *additivity* [4]. It basically means that it is possible to consider independently each observation. The 'only if' direction is best understood when we read the contrapositive: it says that the inconsistence is monotonic (that is, when h is not consistent with some observation set O, it cannot become consistent again when that set grows monotically). In other words, an hypothesis assessed inconsistent on the basis of an observation set cannot become consistent when that set grows. We refer to this latter property, following [4], as *incrementality*. Depending on the application domain, both assumptions may be challenged and relaxed.

We now make more precise our abstract notion of consistency. The basic notion that we shall use is a notion a *group consistency*.

[3] We will denote by $h \sqsubseteq \mathcal{H}$ the fact that h is a conjunction or disjunction of elements of $\mathcal{H}$.

Definition 1 (Group Consistency). *An agent a_i is group consistent wrt. the group of agents G ($GCons(a_i, G)$) iff $Cons(h_i, \cup_{i \in G} O_i)$)*

Note that the definition does not necessarily imply that a_i belongs to G. However, we did not encounter interesting cases where it would be required to consider this case. We shall then assume that $a_i \in G$ in the remainder of this paper.

A stronger notion of consistency requires any agent within the group to be consistent with the entire group.

Definition 2 (Mutual Consistency). *A group of agents is mutually consistent ($MCons(G)$) iff $\forall a_i \in G$, it is the case that $GCons(a_i, G)$.*

Now for the purpose of our work, we shall mainly be interested in some interesting particular cases which depends on the cardinality of the group G:

- *Internal consistency*— this is the limit case when G is limited to a single agent. In this case, Group and Mutual consistency collapse into a single notion that we shall call *internal consistency* ($ICons(a_i)$).
- *Peer consistency*— when the group of agents we consider contains only two agents. In this case we can distinguish both the *peer consistency* of an agent wrt a fellow agent, and the *mutual peer consistency* of a group of two agents (see Fig. 1). This is especially important in our context, since our communication protocols only deal locally with bilateral communications.
- *MAS-consistency* —we conclude with the limit case involving all agents within the society. In this case, we will refer to the *MAS-consistency* of an agent wrt to the society; and to the *mutual MAS-consistency* of a society of agents.

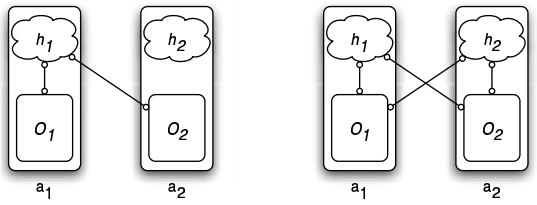


Fig. 1. Peer consistency and mutual peer consistency of agents a_1 and a_2.

For the sake of readability, we now introduce some notational sugar. When we refer to the internal consistency of an agent a_i, we shall simply write $ICons(a_i)$. As for the case of peer and MAS consistency, we will put the cardinality of the group involved as an exponent (recall that the society is composed of n agents). The different notations obtained are summarized in Table 2.2.

	Group Consistency	**Mutual Consistency**
Internal consistency (single agent)	$ICons(a_i)$	$ICons(a_i)$
Peer consistency (pair of agents)	$GCons^2_\alpha(a_i, \{a_i, a_j\})$	$MCons^2_\alpha(\{a_i, a_j\})$
MAS-consistency (society of agents)	$GCons^n_\alpha(a_i, \{a_1, \ldots, a_n\})$	$MCons^n_\alpha(\{a_1, \ldots, a_n\})$

2.3 Revision mechanisms

To ensure its consistency, each agent is equipped with an abstract reasoning machinery that we shall call the *hypothesis formation function* $\mathcal{E}_h$. This function takes a set of observations and an hypothesis as input, and returns a single preferred hypothesis. We assume $h' = \mathcal{E}_h(h, O)$ to be consistent with O by definition of $\mathcal{E}_h$, so using this function on its observation set to determine its favourite hypothesis is a sure way for the agent to achieve consistency. Note however that an hypothesis does not *need* to be generated by $\mathcal{E}_h$ to be consistent with an observation set.

Definition 3 (Individualism). *An agent is said to be* individualistic *iff its working hypothesis h_i may only be modified as a consequence of the application of the hypothesis formation function $\mathcal{E}_h$*

This means that no other agent can directly impose a given hypothesis to it. As a consequence, only a new observation (being it a new perception, or an observation communicated by a fellow agent) can result in a modification of its working hypothesis h_i (but not necessarily of course).

A *internal revision mechanism* is a mechanism μ by which an agent a_i (with its working hypothesis h_i and its observation set O_i) receiving an observation o updates its observation set by adding up o , and update its working hypothesis to $h'_i = \mu(h_i)$.

Definition 4 (guaranteed internal consistency). *An internal revision mechanism μ guarantees internal consistency iff for any agent a_i internally consistent, and any observation o reaching a_i, the application of μ by a_i preserves its internal consistency. More formally: $\forall o \forall O_i \forall h_i, Cons_\alpha(h_i, O_i) \Rightarrow Cons_\alpha(\mu(h_i), O_i \cup \{o\})$.*

A simple internal revision mechanism $\mu_{\mathcal{E}_h}$ consists of replacing h_i by $h'_i = \mathcal{E}_h(h_i, O_i \cup \{o\})$ upon receival of a new observation o. It guarantees internal consistency. In what follows we assume that agents are equipped with a revision mechanism that preserves internal consistency.

A *local revision mechanism* is a mechanism (denoted $\mathcal{M}^2$ following our notation convention) by which an agent a_i receiving an observation o communicates with another agent a_j to update its working hypothesis and possibly the hypothesis of the other agent. A *global revision mechanism* is a mechanism $\mathcal{M}^n$ by which an agent a_i receiving an observation o triggers a series of local revision mechanisms to update its working hypothesis and possibly the hypotheses of the other agents.

Definition 5. *A revision mechanism $\mathcal{M}$ guarantees $GCons(a_i, G)$(resp. $MCons(G)$) iff, for any observation o reaching a_i,it is the case that the execution of $\mathcal{M}$ by a_i with G will result in a situation where $GCons(a_i, G)$ (resp. $MCons(G)$) holds.*
In particular, a local mechanism will be said to guarantee peer consistency (resp. mutual peer consistency), while a global mechanism guarantee MAS-consistency (resp. mutual MAS-consistency).

2.4 Communication

The communication of the agents might be constrained by topological consideration. A given agent will only be able to communicate with a number of *neighbours*. Typically, an agent can only communicate with agents that it knows, but one could imagine topological constraints on communication based on a network of communication links between agents. Who an agent can communicate with will be fixed beforehand, and remain static during the time required for the hypothesis refinement process. We can then construct a communication graph representing these communication links between the agent. (We assume that the relation that links two neighbours in a communication graph is symmetric, but of course not transitive.) A *communication path* will then refer to a path between two agents in the communication graph. In the following, we will suppose that the communication graph is *connected*, that is, there exists a communication path between any pair of agents.

3 Communication Protocols

We are now in a position to examine different (local and global) communication protocols, and to investigate what properties they guarantee, when used with some specific agents' strategies.

3.1 Local revision mechanisms

In this section, we present two different local revision mechanisms. Each of them consists in a communication protocol with an associated strategy.

Unilateral Hypothesis Exchange. The first mechanism $\mathcal{M}_U^2$ uses an asymmetric protocol that we call *Unilateral Hypothesis Exchange* (UHE). The agent applying the mechanism takes an active role in building and refining an hypothesis. It is called the *learner* agent. The second agent is a *critic*, that uses its knowledge to acknowledge or invalidate the proposed hypothesis.

Figure 2 illustrates the protocol. The associated strategy is as follows. The learner agent a_i first updates its hypothesis h_i to h_i' using an internal revision mechanism μ guaranteeing $ICons(a_i)$. Then it *proposes* it to the partner agents a_j, and a_j either replies with $accept_{direct}$ and adopts h_i' as its new working

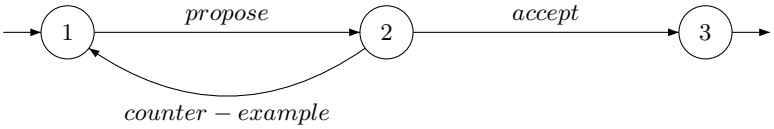


Fig. 2. Unilateral Hypothesis Exchange Protocol (UHE).

hypothesis if $Cons_\alpha(h'_i, O_j)$, or otherwise sends *counter-example(o′)*, where $o' \in O_j$ is such that $Cons_\alpha(h'_i, \{o'\})$ is false. Upon reception of a counter-example, a_i applies again μ to update its hypothesis with the new observation, and propose the resulting hypothesis as before, except that an acceptation will now result in a $accept_{indirect}$ message. (The reason justifying the distinction between $accept_{direct}$ and $accept_{indirect}$ will be become clear later.)

Property 1. When agents are non individualistic $\mathcal{M}^2_U$ guarantees mutual peer consistency.

Proof. Let $n1 = \text{card}(O_j \setminus O_i)$. Each time a_j receives an hypothesis h_i from a_i, it will check the consistence of this hypothesis against its observation set, by considering individually each observation it contains.

- If there is an observation $o \in O_j$ such that $Cons_\alpha(h_i, \{o\})$ is false, then o is sent as a counter-example to a_i. Now suppose that $o \in O_i$, then $Cons_\alpha(h_i, O_i)$ would hold, and by virtue of the incrementality of consistency this would in turn imply that $Cons_\alpha(h_i, \{o\})$, which is known to be false. Therefore $o \notin O_i$, that is $o \in O_j \setminus O_i$. When a_i receives the counter-example, it adds it to its observation set ($O'_i = O_i \cup \{o\}$). This means that n_1 decreases (as $o \notin O_j \setminus O'_i$).
- If there is no observation $o \in O_j$ such that $Cons_\alpha(h_i, \{o\})$ is false (the agent cannot find any counter-example), then, because the $Cons$ relation is complete, it implies that each of its observation, when taken separately, is consistent with h_i. The additivity of the consistency in turn implies that $Cons_\alpha(h_i, O_j)$ (this hypothesis is consistent with the whole observation set of agent a_j). Agent a_j sends an *accept*, which terminates the protocol.

At each hypothesis proposal either n_1 decreases, or the protocol ends with an *accept*. As n_1 is a positive integer, it cannot decreases for ever. So this protocol will end with an *accept* after at most n_1 hypothesis proposals. Upon termination, we necessarily have $Cons_\alpha(h_i, O_j)$ and $Cons_\alpha(h_i, O_i)$ and the adoption of the proposed hypothesis ensures that $h_j = h_i$. That is, agents a_i and a_j each have a common hypothesis that is consistent with O_i and O_j. Hence $MCons(a_i, a_j)$ holds. This demonstrates that this revision mechanisms guarantees mutual peer-consistency. □

After termination, both agents share the *same* mutually consistent hypothesis (which is not required to have mutual peer consistency). However, the fact that

the critic agent adopts the learner's hypothesis clearly violates the property of individualism.

Unilateral Hypothesis Exchange without Adoption. A straightforward possible variant of this mechanism ($\mathcal{M}^2_{UwA}$) preserving the individualism of the agents would require that a_j does not adopt the hypothesis when it accepts it. This variant only guarantees *peer-consistency* instead of *mutual peer-consistency*. The local consistency of a_j with a_i is indeed not ensured, that is, we have $GCons^2_\alpha(a_i, \{a_i, a_j\})$, but not $GCons^2_\alpha(a_j, \{a_j, a_i\})$.

Bilateral Hypothesis Exchange. To get a local revision mechanism that guarantees mutual peer consistency but still preserves the individualism of agents, we symmetricize the unilateral UHE mechanism. The obtained protocol is called the Bilateral Hypotheses Exchange protocol (see Fig. 3). Here, the *critic* agent becomes *learner* agent once it has validated the other agent's hypothesis.

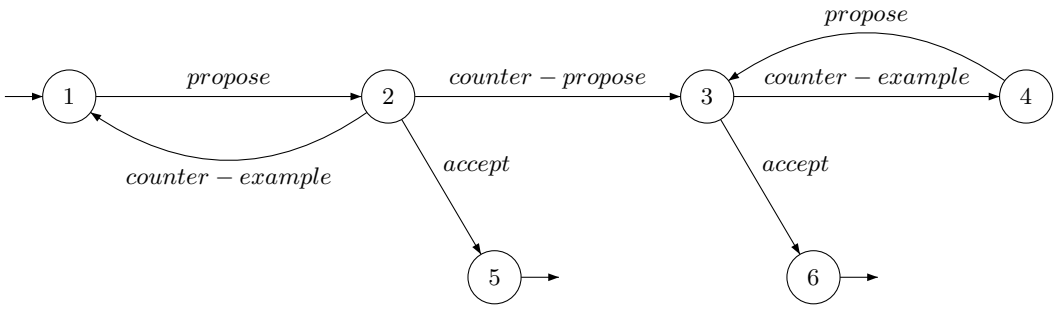


Fig. 3. Bilateral Hypothesis Exchange Protocol (BHE).

Again, we now specify the associated strategy. Upon reception of an hypothesis h_i (by *propose*(h_i)), agent a_j ends up in state 2 and can reply either with an *accept*, a *counter-example*, or a *counter-propose*, as specified by the following strategy:

- If $\exists o \in O_j$ s.t. $Cons_\alpha(h_i, \{o\})$ is false, a *counter-example* consisting of the observation o is sent, as with the UHE mechanism.
- else, we have $Cons_\alpha(h_i, O_j)$ (see proof of Prop. 1). We now want to check that $Cons_\alpha(h_j, O_i)$ holds.
  - If $h_i = h_j$, a_j can directly *accept* h_i.
  - else, it must *counter-propose*(h_j) to invert *critic* and *learner* roles. Then a_i will act as a critic and accept or send counter-examples until consistency is reached.

This mechanism can be interpreted as two reciprocal applications of the unilateral protocol without adoption of hypothesis. Note that when termination is reached via state 6, it is not guaranteed that agents will share the same hypothesis (while it is, by definition of the strategy, when the protocol terminates in state 5). Still, we indeed have $GCons^2_\alpha(a_i, \{a_i, a_j\}$ and $GCons^2_\alpha(a_j, \{a_i, a_j\}$. This mechanism then guarantees *mutual peer-consistency.*

Property 2. $\mathcal{M}^2_B$ guarantees $MCons^2_\alpha(\{a_i, a_j\})$

Proof. Omitted for lack of space, the proof closely follows proof of Prop. 1.

Besides, unlike $\mathcal{M}^2_U$, it respects the agents' individualism. This enables us to get several different consistent hypotheses, but makes it more difficult to maintain the consistency at the global level. Let us now consider this level in more details.

3.2 Global revision mechanisms (for cliques)

Clock-like Hypothesis Propagation. The general idea is to make repeated uses of a local mechanism guaranteeing mutual peer consistency to eventually get a MAS-consistent hypothesis adopted by all agents. The hypothesis must be validated by all agents in turn without being changed. Any change in the hypothesis forces us to check it again from the beginning. Intuitively, as consistence is additive, the hypothesis should grow precise enough to become consistent with all agents.

In more details, the global revision mechanism $\mathcal{M}^n_C$ can be described as follows. When the *learner* agent a_1 applies it after receiving an observation o, a_1 first applies $\mathcal{M}^2_U$ (unilateral hypothesis exchange local revision mechanism) to reach mutual consistency with agent a_2. Then it does the same with agent a_3. If the local protocol ends with an $accept_{direct}$, then a_1 proceeds to exchange its hypothesis with the next agent (a_4), else ($accept_{indirect}$) it goes back to a_2. This iterates, each $accept_{indirect}$ restarting the process with a new hypothesis submitted to a_2. When a_n sends a $accept_{direct}$, it means that the hypothesis has been accepted and adopted in turn by all agents. In such a case, the mechanism ends, and this common hypothesis is MAS-consistent.

Property 3. $\mathcal{M}^n_C$ guarantees mutual MAS-consistence in any clique (fully connected society) of non-individualistic agents.

Proof. Let $n_2 = \text{card}(\bigcup_{i\in\{2,\ldots,n\}} O_i \setminus O_1)$. Using the same principles as the ones used in demonstration of Prop. 1, we show that each time a counter-example is sent (finally resulting in a $accept_{indirect}$) n_2 decreases. As n_2 is a positive integer, it cannot decrase for ever. If n_2 does not decrease, it means that the hypothesis is accepted directly by all agents, that is h_1 is consistent with all the observation sets of the system and all agents have this same hypothesis h_1. Thus, all agents are MAS-consistent, that is, the system is mutually MAS-consistent. This global revision mechanism hence guarantees mutual MAS-consistence. □

Note however that this mechanism requires a_1 to be able to communicate with all other agents. As a_1 can be chosen arbitrarily, this means that this mechanism can only guarantee this property in the context of fully connected societies (cliques).

Clock-like Hypothesis Propagation without Adoption. Similarly to the case of local mechanisms, it is possible to define a simple variant of this mechanism: the mechanism $\mathcal{M}^n_{CwA}$ obtained by using $\mathcal{M}^2_{UwA}$ ($\mathcal{M}^2_U$ without adoption) in $\mathcal{M}^n_C$, respects the individualism of the agents, but can only ensure that the agent applying it is MAS-consistent.

Iterated Clock-like Hypothesis Propagation. We can make it mutually MAS-consistent by making all agents whose hypothesis is inconsistent with o apply it in turn, starting with the agent having received the observation o [4]. We will denote the resulting global revision mechanism by $\mathcal{M}^n_{C*}$, for *Iterated Clock-Like Hypothesis Propagation*. It guarantees *mutual MAS-consistency* in any clique (fully connected society) of possibly individualistic agents.

3.3 Global revision mechanisms (for connected societies)

Static sequential global protocol with propagation When agents are topologically constrained in their communications, a single learner cannot directly propose its hypothesis to all others agents, and the above-mentionned global mechanisms cannot be used. Instead, the global mechanism must rely on some kind of propagation. To ensure that propagation does not reach the same agent from two different ways, the first basic idea is to eliminate cycles by constructing a spanning tree (a tree-like sub-graph that contains all the nodes of the original communication graph).

We describe here a mechanism $\mathcal{M}^n_P$ to construct such a tree while propagating and refining the hypothesis at the same time. The agent receiving a new observation becomes the root of the tree. It begins the propagation by choosing one of its neighbour as its first child, ensuring mutual consistency with it, and asking it to propagate this hypothesis. Figure 4 illustrates the sub-protocol $\mathcal{M}_P(a_i, k)$ triggered when an agent a_i who has already validated its hypothesis with its k first confirmed children, receives the message *propagate(k)*. The associated strategy is described below.

1. If $k = 0$, and a_i has some unchecked neighbours, it will first send *request-link* to each of its unchecked neighbours. These one can answer either *accept-link* (thus becoming a new child of a_i), or *reject-link* if it is already part of this tree. Either way the agents is marked out of the unchecked neighbours, and confirmed sons are somehow ordered.

[4] Recall that the agents whose hypothesis is inconsistent with o will be the only ones that will need to change their hypothesis, since hypotheses consistent with o will remain consistent with $\bigcup_{i\in\{1,\ldots,n\}} O_i \cup \{o\}$ using the compositionality of the consistency relation

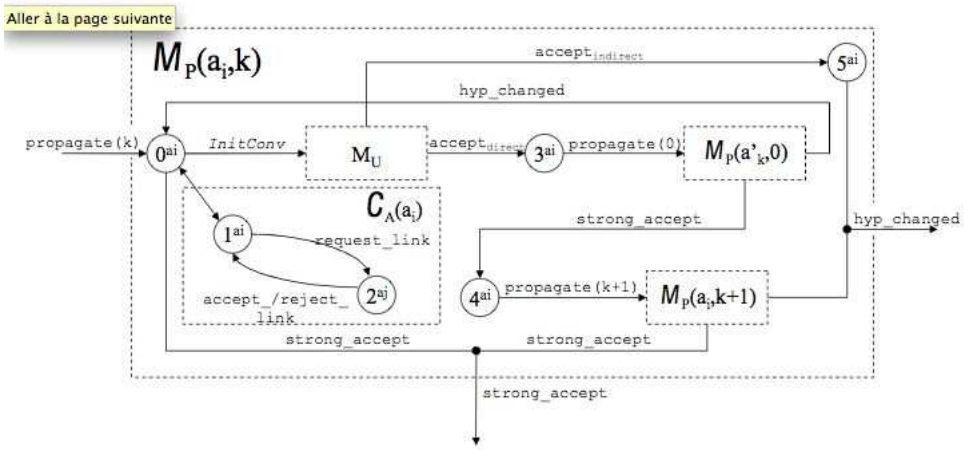


Fig. 4. Global Propagation Mechanism (for agent a_i having already validated its hypothesis with k children.

2. If a_i does not have more than k children (or if $k = 0$ does not have any unchecked neighbour), it means it has no more children to propagate its hypothesis to. It can then confirm this hypothesis to its parent by sending it *strong-accept*. If a_i is the root of the tree, then the mechanism ends there.
3. If a_i has at least $k + 1$ children, it triggers a local revision mechanism $\mathcal{M}_U^2$ with this $(k + 1)^{th}$ child a'_k.
   - If $\mathcal{M}_U^2$ ends with a $accept_{indirect}$, it means that the hypothesis has been changed, and that a_i now has at least one counter-example for its previous hypothesis. As new hypotheses should only be proposed by the root of the tree, a_i sends *hyp-changed* to its parent (or itself if a_i is the root).
   - If $\mathcal{M}_U^2$ ends with a $accept_{direct}$, a_i is now in state 3. It sends *propagate(0)* to a'_k, asking it to propagate in turn the hypothesis to its children, if any, starting with the first. a'_k will then go through the same process that a_i, finally ending with either *strong-accept* or *hyp-changed*.
     - Receiving a *hyp-changed*, a_i would be back in state 1 and initiate a local revision mechanism with the sender to get some counter-example.
     - Receiving a *strong-accept*, a_i would then tries to check its hypothesis with its next children, sending itself a *propagate(k+1)* to iterates the process. *strong-accept* or *hyp-changed* resulting from this would be directed to its parent.

Note that in the case of a clique, $\mathcal{M}_P^n$ is equivalent to $\mathcal{M}_C^n$.

Property 4. In a connected multiagent system of non-individualistic agents, the global revision mechanism $\mathcal{M}_P^n$ guarantees mutual MAS-consistency.

Proof. (Sketch.) First, we consider the linking part of the protocol. The root links to each of its neighbours, and then, in time, each of its children links to any neighbour that is not already linked in the tree. As a results we get a tree containing all agents that are connected to the first agent (the root). If the graph is connected, the whole system is in the resulting tree.

If the depth of the tree (that is the longest path between the roots and one of the leaves) is 1, then the protocol is equivalent to $\mathcal{M}_C^n$ whose properties has been proven.

By recurrence, if the depth of the tree is $d > 1$. Let $n_3 = \text{card}(\bigcup_{i\in\{1,\dots,n\}} O_j \setminus O_r)$ where a_r is the root. Then we can consider each of the children ot the root a_r as a sub-tree containing this child and all its descendants. When this child a_c receives a *propagate* it applies the algorithm to a sub-tree of depth $d-1$, ending with either *strong-accept* or *hyp-changed*. An *hyp-changed* means that a_c has changed its hypothesis $h_c = h_r$. As a parent never adopts an hypothesis from a child, it means that a_c has changed its hypothesis because of a counter-example $o \in \bigcup_{i\in\{1,\dots,n\}} O_j$, which is inconsistent with its old hypothesis h_r. When a_r engage a new conversation with protocol M_U, this counter example o will be added to O_r and n_3 will decrease. Thus only a finite number of *hyp-changed* messages will be send. When all counter-examples are exhausted, each of the child of a_r will finish their propagation by a *strong-accept*, and the protocol will end with the hypothesis h_r being common to all agents and consistent with every observation set of the system.□

3.4 Summary

The following table summarises the different revision mechanisms presented here. The first column is the mechanism name, the second corresponds to the type of revision mechanism (either internal, local or global), the third is the consistency property of the mechanism (with its domain) and the last column indicates if the mechanism respects the individualism of the agents.

Mechanism	**Type**	**Consistency**	**Indiv.**
$\mu_{\mathcal{E}_h}$	internal	internal consistency	yes
$\mathcal{M}_U^2$	local	mutual peer-consistency	no
$\mathcal{M}_{UwA}^2$	local	peer-consistency	yes
$\mathcal{M}_B^2$	local	mutual peer-consistency	yes
$\mathcal{M}_C^n$	global	mutual MAS-consistency (clique only)	no
$\mathcal{M}_{CwA}^n$	global	MAS-consistency (clique only)	yes
$\mathcal{M}_{C*}^n$	global	mutual MAS-consistency (clique only)	yes
$\mathcal{M}_P^n$	global	mutual MAS-consistent (connected societies)	no

4 Applications

4.1 Induction framework

The learning task. We experiment the mechanism proposed above in the case of incremental MAS concept learning.

We consider a propositional language $\mathcal{L}_p$, defined over a set of atoms $\mathcal{A}$. Negative literals are here represented by additional atoms, like $not - a$. The boolean formulae $f = (a \wedge b) \vee (b \wedge \neg c)$ will then be written $(a \wedge b) \vee (b \wedge not - c)$. This representation will be used for learning boolean formulae.

A *hypothesis* will be a disjunction of terms called *prototypes*. Each prototype is a conjunction of atoms $a \in \mathcal{A}$. An *observation* here will be called an *example*. An example is represented by a tag $+$ or $-$ and a description composed of a subset of atoms $e \subseteq A$. The *observation set* of an agent, called here its *example memory* $E = E^+ \cup E^-$, is constituted of a set of positive examples E^+ and a set of negative examples E^-. We shall say that a prototype *covers* an example if its constituting atoms are included in the example, and that a hypothesis covers an example if one of its term covers it.

$Cons_{ind}(h, E)$, the instanciation of $Cons_\alpha$ for this inductive setting, states that h is complete, meaning that it covers all positive examples of E^+, and coherent, meaning that it does not cover any negative example of E^-.

We will described below the incremental learning process, which is internal revision mechanism guaranteeing internal consistence, that we can use as the base internal revision mechanism for local revision mechanism $\mathcal{M}^2_U$ or $\mathcal{M}^2_B$.

Incremental learning process. The learning process is an update mechanism that, given a current hypothesis H, a memory $E = E^+ \cup E^-$ filled with the previously received examples, and a new positive or negative example e, produces a new updated hypothesis. Before this update, the given hypothesis is complete, meaning that it covers all positive examples of E^+, and coherent, meaning that it does not cover any negative example of E^-. After the update, the new hypothesis must be complete and coherent with the new memory state $E \cup \{e\}$. We describe below our single agent update mechanism, inspired from a previous work on incremental learning [7].

In the following, a hypothesis H for the target formula f is a list of terms h, each of them being a conjunction of atoms. H is coherent if all terms h are coherent, and H is complete if each element of E^+ is covered by at least one term h of H. Each term is by construction the *lgg* (least general generalisation) of a subset of positives instances $\{e_1, ..., e_n\}$ [5], that is the most specific term covering $\{e_1, ..., e_n\}$. The *lgg* operator is defined by considering examples as terms, so we denote as $lgg(e)$ the most specific term that covers e, and as $lgg(h, e)$ the most specific term which is more general than h and that covers e. Restricting the term to *lgg* is the basis of a lot of Bottom-Up learning algorithms (for instance [5]). In the typology proposed by [9], our update mechanism is an *incremental learner with full instance memory*: learning is made by successive updates and all examples are stored. Note that this is also a case of learning from interpretations [3].

The update mechanism depends of the ongoing hypothesis H, the ongoing examples E^+ and E^-, and the new example e. There are three possible cases:

- e is positive and H covers e, or e is negative and H does not cover e. No update is needed, H is already complete and coherent with $E \cup \{e\}$.

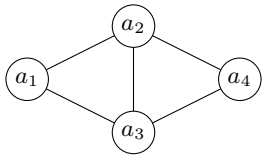
- e is positive and H does not cover e: e is denoted as a *positive counterexample* of H. Then we seek to generalise in turn the terms h of H. As soon as a correct generalisation $h' = lgg(h, e)$ is found, h' replaces h in H. If there is a term that is less general that h', it is discarded. If no generalisation is correct (meaning here coherent), $H \cup lgg(e)$ replaces H.
- e is negative and H covers e: e is denoted as a *negative counterexample* of H. Each term h covering e is then discarded from H and replaced by a set of terms $\{h'_1, ..., h'_n\}$ that is, as a whole, coherent with $E^- \cup \{e\}$ and that covers the examples of E^+ uncovered by $H - \{h\}$. Terms of the final hypothesis H that are less general than others are discarded from H.

Note that this mechanism tends to both make a minimal update of the current hypothesis and minimise the number of terms in the hypothesis, in particular by discarding terms less general than other ones after updating a hypothesis.

Mapping summary. The following table summarizes the correspondence between our formal model and the inductive framework.

Formal Model	**Inductive framework**
Supporting language	propositional ($\mathcal{L}_P$) over a set of atoms $\mathcal{A}$
Possible hypotheses $\mathcal{H}$	any prototype p (conjunction of atoms)
Belief base (A_i)	set of prototype $\{p_1, \ldots, p_m\}$
Working hypothesis (h_i)	disjunction of prototype in A_i (h_i)
Observations ($\mathcal{O}$)	Examples : label + description (set of atoms)
Observation set (O_i)	Example memory $E_i = E_i^+ \cup E_i^-$
Prior common knowledge ($\mathcal{F}$)	none
Consistency relation ($Cons_\alpha(h, O)$)	completeness and coherence ($Cons_{ind}(h, E)$)
Internal revision mechanism (μ or $\mathcal{E}_h$)	inc. bottom-up learning process ($h' = \mu(E, h, e)$)

Example. We give here an example of propagation with 4 agents, connected as shown below.

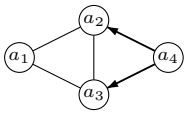


The propositional language has an atom for each letter in the alphabet. Examples are set of letter (words, but without order nor redundancy). Positives examples are examples that contains either a or oc. We consider the system after several learning turn. The example memory of the agents are the following:
$E_1 = \{hat+\}$
$E_2 = \{boc+\}$
$E_3 = \{rot-, bat+\}$
$E_4 = \{\ \}$
The previous deliberation has lead the system to adopt the common hypothesis $h = h_1 = h_2 = h_3 = h_4 = at \vee boc$.

Now, agent a_4 receives a new example $toc+$, that is inconsistent with h. He triggers the global revision mecanism $\mathcal{M}_P$ as detailed below:

$\mathcal{M}_P(a_4, 0)$

$\mathcal{C}_A(a_4)$
4→2 request-link
2→4 accept-link
4→3 request-link
3→4 accept-link



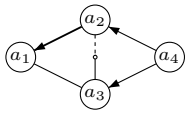
$M_U(a_4, a_2)$
4→2 propose(t∨boc)
2→4 accept
OUT: 2→4 $accept_{direct}$

Post
$h_4 = t \vee boc$ unchanged
E_4 unchanged

4→2 propagate(0)

$\mathcal{M}_P(a_2, 0)$

$\mathcal{C}_A(a_2)$
2→3 request-link
3→2 reject-link
2→1 request-link
1→2 accept-link



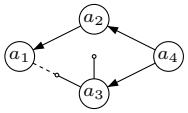
$M_U(a_2, a_1)$
2→1 propose(t∨boc)
1→2 accept
OUT: 1→2 $accept_{direct}$

Post
$h_2 = t \vee boc$ unchanged
E_2 unchanged

2→1 propagate(0)

$\mathcal{M}_P(a_1, 0)$

$\mathcal{C}_A(a_1)$
1→3 request-link
3→1 reject-link



OUT: 1→2 strong-accept

2→2 propagate(1)

$\mathcal{M}_P(a_2, 1)$
OUT: 2→4 strong-accept

OUT: 2→4 strong-accept

4→4 propagate(1)

$\mathcal{M}_P(a_4, 1)$

$M_U(a_4, a_3)$
4→3 propose($t \vee boc$)
3→4 contre-exemple(rot)
4→3 propose(oc)
3→4 contre-exemple(bat)
4→3 propose($oc \vee bat$)
3→4 accepte
OUT: 3→4 $accept_{indirect}$

Post
$h_4 = oc \vee bat$ changed
$E_4 = E_4 \cup \{rot-, bat+\}$

OUT: 4→4 hyp-changed

OUT: 4→4 hyp-changed

4→4 propagate(0)

$\mathcal{M}_P(a_4, 0)$

$M_U(a_4, a_2)$
4→2 propose(oc∨bat)
2→4 accept
OUT: 2→4 $accept_{direct}$

$Post$
$h_4 = oc \vee bat$ unchanged
E_4 unchanged

4→2 propagate(0)

$\mathcal{M}_P(a_2, 0)$

$M_U(a_2, a_1)$
2→1 propose(oc∨bat)
1→2 contre-exemple(hat)
2→1 propose(oc∨at)
1→2 accept
OUT: 1→2 $accept_{indirect}$

$Post$
$h_2 = oc \vee at$ changed
$E_2 = E_2 \cup \{hat+\}$

OUT: 2→4 hyp-changed

$M_U(a_4, a_2)$
4→2 propose(oc∨bat)
2→4 contre-exemple(hat)
4→2 propose(oc∨at)
2→4 accept
OUT: 2→4 $accept_{indirect}$

$Post$
$h_4 = oc \vee at$ changed
$E_4 = E_4 \cup \{hat+\}$

OUT: 4→4 hyp-changed

4→4 propagate(0)

$\mathcal{M}_P(a_4, 0)$

$M_U(a_4, a_2)$
4→2 propose(oc∨at)
2→4 accept
OUT: 2→4 $accept_{direct}$

$Post$
$h_4 = oc \vee at$ unchanged
E_4 unchanged

4→2 propagate(0)

$\mathcal{M}_P(a_2, 0)$

$M_U(a_2, a_1)$
2→1 propose(oc∨at)
1→2 accept
OUT: 1→2 $accept_{direct}$

$Post$
$h_2 = oc \vee at$ unchanged
E_2 unchanged

2→1 propagate(0)

$\mathcal{M}_P(a_1, 0)$
OUT: 1→2 strong-accept

2→2 propagate(1)

$\mathcal{M}_P(a_2, 1)$
OUT: 2→2 strong-accept

OUT: 2→4 strong-accept

4→4 propagate(1)

$\mathcal{M}_P(a_4, 1)$

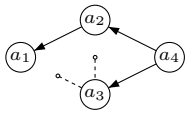
$M_U(a_4, a_3)$
4→3 propose(oc∨at)
3→4 accept
OUT: 3→4 $accept_{direct}$

$Post$
$h_4 = oc \vee at$ unchanged
E_4 unchanged

4→3 propagate(0)

$\mathcal{M}_P(a_3, 0)$

$\mathcal{C}_A(a_3)$
3→2 request-link
2→3 reject-link
3→1 request-link
1→3 reject-link



OUT: 3→4 strong-accept

4→4 propagate(2)

$\mathcal{M}_P(a_4, 2)$
OUT: 4→4 strong-accept

OUT: 4→4 strong-accept

OUT: 4→4 strong-accept

4.2 Abduction framework

Formal machinery. We present here the adaptation of our formal model to an abductive framework, inspired from Poole's Theorist system. We consider a first order language $\mathcal{L}_1$. $\mathcal{O}$ stands for the predefined set of possible observations that can possibly be made in any instance of the described system. (In representing this set, we shall use non-ground schemes where any term with a capital letter denotes a variable, and all variables are universally quantified. It then represents all its ground instances in a given Herbrand universe.) Each agent a_i is modeled as a slightly modified version of an instance of a Theorist system [11]:

$$\langle \mathcal{F}, \mathcal{H}, \preceq, O_i, \Theta_i, h_i \rangle$$

where

- $\mathcal{F}$ a set of *facts*, closed formulae taken as being true in the domain;
- $\mathcal{H}$ a set of *abducible predicates* which act as *conjectures*, possible hypotheses common to all agents;
- $\preceq$ is the *preference relation*, a (complete) pre-order on the hypotheses that we assume common to all agents.
- $O_i \subseteq \mathcal{O}$ is a set of grounded formulae representing the *observations* made so far by the agent. Each agent knows every observation in this set to be true, as its sensors are considered perfect;
- Θ_i is the set of *selected hypotheses*, which will be defined below;
- h_i is the *favourite hypothesis* of the agent, that it assumes to be true, and that serves as a basis for deriving its belief set.

$\mathcal{F}, \mathcal{H}, \preceq$ are common to all agents. $\mathcal{F}$ represents prior knowledge of the agent on the environment and its rules. We first recall a number of basic definitions, used for defining Θ_i, from which h_i is taken.

Definition 6 (Scenario [11]). *A scenario of $(\mathcal{F}, \mathcal{H})$ is a set θ of ground instances of elements of $\mathcal{H}$ such that $\theta \cup \mathcal{F}$ is consistent (that is $\theta \cup \mathcal{F} \not\models \Box$).*

Definition 7 (Explanation of a closed formulae [11]). *If g is a closed formula, then an explanation of g from $(\mathcal{F}, \mathcal{H})$ is a scenario θ of $(\mathcal{F}, \mathcal{H})$ that (with $\mathcal{F}$) implies g (that is $\theta \cup \mathcal{F} \models g$).*

We now introduce a couple of further notions that proved to be appropriate in our context. Events occurring in the world and observed by the agents may or may not be explained, or contradicted, by the agent model.

Definition 8 (Positive observation). *A positive observation of $(\mathcal{F}, \mathcal{H})$ is an observation $o \in \mathcal{O}$ such that there exists an explanation of o from $(\mathcal{F}, \mathcal{H})$*

Definition 9 (Negative observation). *A negative observation of $(\mathcal{F}, \mathcal{H})$ is an observation $o \in \mathcal{O}$ such that there exists an explanation of $\neg o$ from $(\mathcal{F}, \mathcal{H})$*

In the following, we shall note $P(O)$ to refer to the set of all positive observations of $(\mathcal{F}, \mathcal{H})$ in $O \subseteq \mathcal{O}$, and $N(O)$ to refer to the set of all negative observations of $(\mathcal{F}, \mathcal{H})$ in $O \subseteq \mathcal{O}$. Note that this is not necessarily a partition: some observations may have no explanation, while some others may be explained and have at the same time their negation explained. Moreover, if $o \in \mathcal{O}$ and $\neg o \in \mathcal{O}$, then o positive iff $\neg o$ negative, and reciprocally.

Definition 10 (Explanation of an observation set). *If $O \subseteq \mathcal{O}$ is a set of observations, an explanation of O from $(\mathcal{F}, \mathcal{H})$ is an explanation θ of $P(O)$ from $(\mathcal{F}, \mathcal{H})$ such that $\theta \cup \mathcal{F} \cup N(O)$ is consistent (which implies the consistency of $\theta \cup O$). That is: $\theta \cup \mathcal{F} \models P(O)$ and $\theta \cup \mathcal{F} \cup N(O) \not\models \square$.*

In the following, we shall also refer to the conjunction h of the elements of θ as the *hypothesis* associated to this explanation.

Definition 11 (Irredundant explanation). *An irredundant explanation of O from $(\mathcal{F}, \mathcal{H})$ is an explanation such that if any element of its associated hypothesis set θ is removed from it, it is no longer an explanation of O. In other words, an irredundant explanation is an explanation that is minimal wrt. set inclusion.*

Typically, as suggested by the aforementioned model, different explanations will exist for a given formula. What should be the preference relation between explanations? Clearly there can be many different ways to classify preferred explanations. In our framework, we shall use three classical options (the two first being borrowed to [11], and the last one being trivially based on cardinality).

1. *minimal explanation*— prefer the explanations that make the fewest (in terms of set inclusion) assumptions. This is what we have defined as *irredundant explanation* to avoid confusion with the minimal explanations according to the preference relation, borrowing a term from set-cover based abduction [10];
2. *least presumptive explanation*— an explanation is less presumptive than another explanation if it makes fewer assumptions in terms of what can be implied from this explanation. An explanation ξ_1 is less presumptive that another explanation ξ_2 iff $\forall g$ s.t. $\xi_1 \models g$, it is the case that $\xi_2 \models g$. Therefore, a least presumptive explanation is an explanation which is not less presumptive than any other explanation.
3. *minimal cardinality*— is self-explanatory, and prefers explanations that contains the fewest terms as possible.

Based on this system, we can now define precise the definition of Θ_i and h_i:

– Θ_i, the set of *selected hypotheses* associated with each irredundant explanations of the observation set O_i. $\Theta_i = \{\bigwedge_{p \in \theta} p | \theta$ is an irredundant explanation of O_i from $(\mathcal{F}, \mathcal{H})$ $\}$. For a given set of observation O_i, $\mathcal{E}_\Theta$, the *explanation function* returns the set of all hypotheses associated with an irredundant explanation of O_i from $(\mathcal{F}, \mathcal{H})$.

– h_i is the *favourite hypothesis* used as a *working hypothesis*. It is an element of Θ_i chosen by the agent among minimal hypothesis according to the preference relation $\preceq_p$, that is $h_i \in min_{\preceq_p}(\Theta_i)$. We can then abstract away all these elements in the *hypothesis formation function*: $\mathcal{E}_h$ associates any set of observation O_i with an hypothesis $h_i \in min_{\preceq_p}(\mathcal{E}_\Theta(O_i))$.

The instanciation of the *consistency relation* for this abductive setting $Cons_{abd}(h, O)$ is then defined as the combination of two properties:

– *coherence*, that is $\forall o \in N(O), \{h\} \cup \mathcal{F} \not\models \neg o$ (which implies $\forall o \in O, \{h\} \cup \mathcal{F} \not\models \neg o$)
– *completeness*, that is $\forall o \in P(O), \{h\} \cup \mathcal{F} \models o$.

Note that this definition of consistence corresponds to the logical definition of abductive explanation. We have: $Cons_{abd}(h, O)$ iff h is an abductive explanation of O for the theory $\mathcal{F}$. Our definition of $\mathcal{E}_h$ ensures these properties, and so enables each agent to guarantee its internal consistence.

Example. We consider several agents trying to diagnose together which disease affects some person. They share the same medical knowledge $\mathcal{F}$ about diseases and symptoms, but each of the agent has only observed some incomplete part of the symptoms affecting the person. Observations would be the observable symptoms (effects or manifestations of the diseases), abducible predicates correspond to diseases (causes of the symptoms), and the facts are the rules linking diseases and symptoms. Hypotheses could then be conjunction of abducible predicates.

$$\begin{aligned}\mathcal{O} = \{\ & Fever(X), Cough(X), Mucus(X), ThroatAche(X), SwallowsEasily(X),\\ & \neg Fever(X), \neg Cough(X), \neg Mucus(X), \neg ThroatAche(X), \neg SwallowsEasily(X)\}\\ \mathcal{H} = \{\ & Flu(X), Bronchitis(X), Angina(X), HayFever(X)\}\end{aligned}$$

and

$$\begin{aligned}\mathcal{F} = \{\ & Flu(X) \Rightarrow Fever(X),\\ & Flu(X) \Rightarrow Cough(X),\\ & Bronchitis(X) \Rightarrow Cough(X),\\ & Bronchitis(X) \Rightarrow Mucus(X),\\ & Bronchitis(X) \Rightarrow ThroatAche(X),\\ & Angina(X) \Rightarrow Fever(X),\\ & Angina(X) \Rightarrow ThroatAche(X),\\ & Angina(X) \Rightarrow \neg SwallowsEasily(X),\\ & HayFever(X) \Rightarrow Mucus(X)\}\end{aligned}$$

Note that $Fever(X), Cough(X), Mucus(X)$ and $ThroatAche(X)$ and $\neg SwallowsEasily(X)$ are positive observation schemes, whereas $SwallowsEasily(X)$, $\neg Fever(X)$, $\neg Cough(X)$, $\neg Mucus(X)$, and $\neg ThroatAche(X)$ are negative observation schemes.

Now suppose that agent a_1 has observed $O_1 = \{Fever(Tom), \neg ThroatAche(Tom)\}$.

From these observations, it can deduce two explanations : $\theta_u = \{Flu(Tom)\}$ and $\theta_{uh} = \{Flu(Tom), HayFever(Tom)\}$. But θ_{uh} is not irredundant (as $\theta_u \subset \theta_{uh}$ is also an explanation). Therefore, $\Theta_1 = \{Flu(Tom)\}$ and $h_1 = Flu(Tom)$.

Suppose in addition that agent a_2 has observed $O_2 = \{Cough(Tom), Mucus(Tom)\}$.
From these observations, it can deduce two explanations : $\theta_b = \{Bronchitis(Tom)\}$ and $\theta_{uh} = \{Flu(Tom), HayFever(Tom)\}$, as well as other redundant explanation such has $\{Flu(Tom), HayFever(Tom), Angina(Tom)\}$, etc. Therefore, $\Theta_2 = \{Bronchitis(Tom), Flu(Tom) \wedge HayFever(Tom)\}$.
Our preference relation prefers explanations with minimal cardinality (as it seems more likely that someone has one disease rather that several). It thus selects $h_2 = Bronchitis(Tom)$.

Now if these two agents were to communicate using the Bilateral Hypothesis Exchange Protocol, we would get the following dialogue:

a_1 sends to a_2 *propose*$(Flu(Tom))$
a_2 sends to a_1 *counter-example*$(Mucus(Tom))$
a_1 sends to a_2 *propose*$(Flu(Tom) \wedge HayFever(Tom))$
a_2 sends to a_1 *counter-propose*$(Bronchitis(Tom))$
a_1 sends to a_2 *counter-example*$(\neg ThroatAche(Tom))$
a_2 sends to a_1 *propose*$(Flu(Tom) \wedge HayFever(Tom))$
a_1 sends to a_2 *accept*

Note that in this case, the agents were not consistent before receiving a single new information. Hypothesis Exchange protocols can then restore consistency between two agents even if each of them did receive several new observations since they were consistent.

Mapping summary. We report in the following table the correspondence between our formal model and the abductive framework used here.

Formal Model	**Abductive framework**
Supporting language	first order logic ($\mathcal{L}_1$)
Possible hypotheses $\mathcal{H}$	set of abducible predicates $\mathcal{H}$
Belief base (A_i)	favourite explanation of O_i from $(\mathcal{F}, \mathcal{H})$
Working hypothesis (h_i)	conjunction of grounded abducibles predicates of A_i (h_i)
Observations ($\mathcal{O}$)	set of possible observations, grounded formulae ($\mathcal{O}$)
Observation set (O_i)	observation set O_i, contains $P(O_i)$ and $N(O_i)$
Prior common knowledge ($\mathcal{F}$)	set of facts ($\mathcal{F}$)
Consistency relation ($Cons_\alpha(h, O)$)	compl. for $P(O)$, coherence with $N(O)$ ($Cons_{abd}(h, O)$)
Internal revision mechanism (μ or $\mathcal{E}_h$)	Theorist based hyp. formation func. ($h' = \mathcal{E}_h(O)$)

5 Related Works

This paper extends our previous work by setting up an abstract framework for hypothesis refinement under communication constraints, that can be instantiated with abductive or inductive reasoning agents. More specifically, [2] was concerned with a specific dynamic abductive application where different types of mechanisms were experimented, while [2] mainly experimented one specific mechanism in the context of distributed learning. This paper abstract away from these applications and proposes a detailed catalogue of generic mechanisms for hypothesis refinement. Now if we consider the specific contexts of abductive or inductive reasoning, some interesting links can be made with other works.

The problem of multiagent diagnosis has been studied by Roos and colleagues [12], where a number of distributed entities try to come up with a satisfying global diagnosis of the whole system. They show in particular that the number of messages required to establish this global diagnosis is bound to be prohibitive, unless the communication is enhanced with some suitable protocol. This approach, however, does not consider any specific constraint governing agents' interactions. One other piece of work connected to the trend of agent-oriented computational logic that seems particularly relevant to our approach is that of Gavanelli et al. [6]. They identify under which circumstances some notion of global or local consistency (the counterparts of our mutual MAS and internal consistency) can be achieved. However, they abstract away from any specific protocol in their study, concentrating on the definition of what it semantically means for a group to reason abductively. There is also a growing body of work related to distributed learning, but as far as we know, none of these works are specifically oriented on the design of suitable protocols for hypothesis refinement.

6 Conclusion

The main objective of this paper has been to set up the foundations of a framework allowing the study of the properties of different protocols for hypothesis refinement. We have investigated how (under what conditions) convergence to a consistent state can be guaranteed at global levels: (i) between two agents; (ii) in a clique of agents; and (iii) in general in a society of agents. In particular, we have shown how this approach can be instantiated in the context of abductive or inductive tasks; and provided different examples illustrating our approach. We regard this problem as a crucial element for the development of applications involving a distributed knowledge.

Now the concrete assessment of these protocols is very much dependent of the actual domain where they are eventually implemented. In a first companion paper [2], those aspects have been experimentally investigated in the context of incremental inductive supervised concept learning. Especially the effectiveness in term of accuracy and efficiency in term of redundancy were evaluated for protocol $\mathcal{M}_C^n$. In a second companion paper [1], the efficiency and effectiveness of

such protocols have been studied in the very different context of a critical situation (hence involving a dynamic communication network) involving a number of agents aiming at escaping from a burning building. In this abductive application, agents observe the evolving environment and communicate observations and hypothesis. The results reported in these papers indeed emphasize the importance of the application domain for protocol design. We observed for instance in this last application that the efficiency of hypothesis exchange (as opposed to a mere observation exchange) depends very much on the type of map representing the building. There are of course many ways to elaborate on the proposal put forward in this paper. On the longer term, one very significant but highly challenging departure from the framework developed here would be to relax the assumption of compositionality of the consistency relation.

Acknowledgments. We would like to thank Henry Soldano whose detailed comments helped to greatly improved the paper.

References

1. Gauvain Bourgne, Gael Hette, Nicolas Maudet, and Suzanne Pinson. Hypotheses refinement under topological communication constraints. In *Proceedings of AAMAS-2007*. ACM Press, 2007.
2. Gauvain Bourgne, Amal El Fallah Seghrouchni, and Henry Soldano. Smile : Sound multi-agent incremental learning ;-). In *Proceedings of AAMAS-2007*. ACM Press, 2007.
3. Luc De Raedt. Logical settings for concept-learning. *Artif. Intell.*, 95(1):187–201, August 1997.
4. Peter Flach. Abduction and induction: Syllogistic and inferential perspectives. In *Proceedings of the Workshop on Abductive and Inductive Reasoning*, 1996.
5. Johannes Fürnkranz. A pathology of bottom-up hill-climbing in inductive rule learning. In *ALT*, volume 2533 of *LNCS*, pages 263–277. Springer, 2002.
6. Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. An abductive framework for information sharing in multi-agent systems. In *Proceedings of the 4th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-IV)*, LNAI 3259, pages 34–52. Springer, 2004.
7. M'hammed Henniche. MGI: an incremental bottom-up algorithm. In *IEEE Aust. and New Zealand Conference on Intelligent Information Systems*, pages 347–351, 1994.
8. Sébastien Konieczny and Ramón Pino Pérez. Propositional belief base merging or how to merge beliefs/goals coming from several sources and some links with social choice theory. *European Journal of Operational Research*, 160(3):785–802, 2005.
9. Marcus A. Maloof and Ryszard S. Michalski. Incremental learning with partial instance memory. *Artificial Intelligence*, 154(1-2):95–126, 2004.
10. Y. Peng and J. Reggia. *Abductive Inference Models for Diagnostic Problem Solving*. Springer Verlag, 1990.
11. D. Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110, 1989.
12. N. Roos, A. ten Tije, and C. Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *Proceedings of AAMAS03*, pages 655–661, 2003.

Logic-statistic modeling and analysis of biological sequence data: *A Research Agenda*

Henning Christiansen

Research group PLIS: Programming, Logic and Intelligent Systems
Department of Communication, Business and Information Technologies
Roskilde University, P.O.Box 260, DK-4000 Roskilde, Denmark
E-mail: henning@ruc.dk

Abstract. We describe here the intentions and plans of a newly started, funded research project in order to further the dialogue with the international research in the field. The purpose is to obtain experiences for realistic applications of flexible and powerful modeling tools that integrate logic and statistics, as exemplified by the PRISM system. As part of this, we will develop systematic and automatic optimizations, and the overall goal is to see how far it is possible to promote such techniques in computational biology.

1 Introduction

We describe the background and intentions of a newly started, funded research project. The primary project goal is to promote the application of logic-statistic modelling tools for the analysis of biological sequence data, where we use the PRISM system developed by Sato and Kameya [1, 2] as a starting point. Compared with traditional models based on HMMs and SCFGs, the techniques in consideration lift the expressive power to, in principle, Turing-complete languages. However, the price of the additional expressibility is comptuational complexity. In this project we will approach these computational problems with state-of-the-art program analysis and transformation techniques, including to see how existing and optimized implementations for traditional models can be integrated. This research is combined with investigations of biological problems which in themselves are relevant and provide good test cases. The project group include researchers in Computer Science and in Biochemistry from a number of unversities as well as industrial partners representing a major supplier of probiotic products for the dietary supplement industry, Chr. Hansen, plus a leading supplier of bioinformatics software, CLC bio; the project runs 2007–2011 and includes funding for several PhDs and postdocs, plus workshops etc.

There is an emerging trend in applying logic-based learning techniques for computational biology that we shall not review here, and in the present project we are especially interested in technologies based on logic programming as the central modeling paradigm. As is well known, logic programming is superior for modeling many linguistic phenomena due to elegance and flexibility, but not always with respect to efficiency.

We consider the abductive potential in such techniques. An overall model of the relationship between biological properties (understood in a very wide sense) and its encoding in a sequence (a protein, a genome, ...) is described as a model expressed as a statistic-logic program; known and already investigated sequences are used for training the model, i.e., for learning probabilities for its random variables; and with the learned probabilities, the model can be used for prediction of the "best" proposal for the biological properties encoded in a given sequence. This is quite analogous to the paradigm of linguistic discourse analysis as abduction [3] which we have studied in a logic programming context [4].

As test cases, we may use eukariotic problems from the literature for pressing complexity, but the main biological applications reflect the project group's expertise and practical needs in prokaryotic biochemistry. This include gene finding in health promoting bacteria, phylogenetic gene prediction, and prediction of gene function.

In section 2, we give an introduction to PRISM by a biologically inspired example which leads us to some observations why these techniques may be suited for biological sequence analysis. Section 3 describes our first nontrivial application for sequence data, which, however, does not involve prediction at its present state. Section 4 indicates some the computational problems that we intend to approach, and section 5 suggest possible extensions to the present PRISM, and section 6 provides a conclusion and discussion of a few specific points.

2 Working with sequence data in PRISM

Syntactically, PRISM extends the Prolog language with discrete, random variables, called `msw` for *multi-valued switch.* The following declarations, which are part of the model we describe below, define two singleton variables and two parameterized, and in principles, infinite classes of variables.

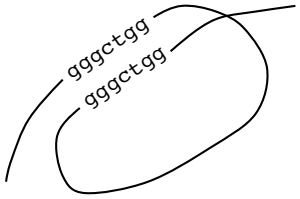
```
values(moreInBetween,[stop,continue]).
values(moreGlue,[stop,continue]).
values(which(_),[a,c,g,t]).
values(which(_,_),[a,c,g,t]).
```

Executing a call `msw(moreInBetween,X)` within a PRISM program will, by a random choice, assign one of `X=stop` or `X=continue`. As shown by [5], Prolog's traditional Herbrand model semantics generalizes immediately to a probabilistic semantics when probabilities are given for each random variable (provided that a few restrictions are respected on how `msw` is used in the program). In other words, such a PRISM program is a probabilistic model in the sense that it provides a probability distribution for all events that can be formulated in the program's logical language.

The following example program models a simple loop structure in a sequence; it is inspired by protein folding but apart from that it is completely artificial and has nothing to do with biology. If a sequence has the following form,

```
---------gggctgg-----------------------------gggctgg------
```

where a "-" indicates a random letter, a loop structure can arise assuming that each letter tend to glue to an identical letter as indicated by the following drawing.



Let us arbitrarily assume that the actual gluing zones can be relevantly described by a first order Markov model, and the stuff in between by a second order; in other words, we indicated that the two kinds of substrings are characterized by their own distribution of letters and other regularities. The following program defines a model for such sequences which captures these assumptions. Notice that random variables are used to generate the first copy of the gluing string, which then is repeated as a verbatim copy. Random variables `moreInBetween` and `moreGlue` determine the length of each subsequence. A goal `sequence(`G`,`S`)` means that sequence S has a loop glued together by the substring G.

```
sequence(G,S):-
    inBetween(-,S,S1),
    glue(G,-,-, S1,S2),
    inBetween(-,S2,S3),
    glueCopy(G, S3,S4),
    inBetween(-,S4,[]).

inBetween(L,S1,S2):- msw(moreInBetween,YN), inBetween2(L,S1,S2,YN).
inBetween2(_,S,S,stop).
inBetween2(L1,[L|S1],S2, continue):- msw(which(L1),L), inBetween(L,S1,S2).

glue(G,L1,L2,S1,S2):- msw(moreGlue,YN), glue2(G,L1,L2,S1,S2,YN).
glue2([],_,_,S,S,stop).
glue2([L|G],L1,L2,[L|S1],S2, continue):-
   msw(which(L1,L2),B), glue(G,L2,L,S1,S2).

glueCopy([],S,S).
glueCopy([L|G],[L|S1],S2):- glueCopy(G,S1,S2).
```

With fixed probabilities, this program can generate sets of samples which reflect the probabilistic semantics. PRISM can also learn probabilities from files of samples, which we can imagine represent sequences whose actual structure has been recorded in the laboratory. Assuming that the model is really appropriate for the sequences in questions, the detailed probabilities learned express regularities or structures within the substrings that may be new for the biological researcher, and in this way there is a flavour of induction. We can assume a file of training date as follows. The dominance of letters `g` and `c` in the gluing strings holds throughout the file; the use of bold face is for emphasis only and not part of the file.

```
sequence([g,g,g,c,t,g,g],[a,g,g,g,c,t,g,g,a,a,t,c,a,a,a,t,c,t,
        t,t,a,a,c,g,g,g,c,t,g,g,a,g,a,c,t,a,t,g,t,t,a,g,a,a,a,a]).
sequence(......, ......).
sequence(...., .....).
...
```

With the probabilities learned from these samples, PRISM can do Viterbi computations in order to predict, for a given sequence, the most likely glue substring out of the "logically" possible; this process can be compared with probabilistic abduction. See the following example, where the preferred gluing string (out of several possible) has similar high frequencies of `g` and `c` as the training data.

```
?- viterbig(sequence(G,[t,a,t,a,g,c,g,c,t,a,t,a,g,c,g,c,t,a,t,a])).
G = [g,c,g,c]
```

While being extremely simplistic, this example illustrates the qualities of PRISM that has motivated us to take it as initial platform for our research project:

- The clean semantics of the system ensures the we can accept a program as a formalized model of a phenomenon.
- Known probabilistic models, e.g., HMM, SCFG, Bayes' networks, etc. can be expressed in straightforward ways and, not least, *combined* while still maintaining a coherent model.
- Arbitrary data structures and auxiliary variables can be introduced whenever convenient without destroying the clean semantics.
- Compared with other linguistically oriented models, PRISM lifts to in principle Turing-complete languages (as opposed to regular (HMM) or context-free (SCFG)); the model above is based on a non-context-free language.
- The same model (= PRISM program) can be used for learning, sample generation, and prediction which gives a theoretically sound basis for comparing the different phases.[1]

3 Logic-statistic modeling for testing gene finders

Here we describe what seems to be the first non-trivial application of PRISM for a biological sequence problem. In a recent paper [6], we describe a partial model for genome sequences, which is used for the problem of producing artificial test data for testing existing gene finder programs; this problem is relevant since production of authentic test data is very expensive and sparsely available. The idea of using artificial test data for this purpose has been applied before [7, 8]. Basically, these works applied HMMs combined with ad hoc principles and it can be questioned [6] how "natural" the produced date are. Our aim of using PRISM is, of course, to provide a more detailed and reliable model, and thus better test data.

[1] Occationally it may be needed to use a few low-level Prolog hacks to have the program perform efficiently for the different phases, but done in a disciplined way, this may not destroy the semantics.

The model is defined for the intergenic subsequences capturing GC islands, occurrences of various sorts of repeat strings, and mutations. Probabilities were learned from the a set of sequences that were marked up using existing mapping tools. Since a full model was not available which also covers genes, we pasted together artificially produced intergenic subsequences with authentic genes. The conclusion from testing three different gene finders was that they all predicted too many genes and different genes. We shall not go into details here with this model, but only mention some distinct features of it that shows the flexibility a modeling tool which is based on a general and elegant programming language such as Prolog.

- Using PRISM's parameterized random variables, we defined a multi-level model, the top-level describing GC island, and embedding it in an abstract data type, each random variable and probability defined for the lower levels, exists in two versions for being in or outside of a GC island. The remaining part is a two-level Markov model, one level determining the alternation of different substring types (different sorts repeat strings, plain coloured noise, etc.) and the lowest level how these substring are built from letters.
  — This is a quite complicated model, but which by an experienced logic programmer can be organized in a readable way.
- A fairly large catalogue of named repeater strings were added, encoded from existing resources as a binary predicate; still this respects a reasonable semantics.
- To reduce complexity in handling mutations, we added a preprocessing phase for the training data, which uses a best match algorithm to produce exactly one detailed mark-up of the mutations that are supposed to have taken place when a (section of) a repeater string was copied into the sequence.
- Using a 64 bit version of PRISM we could, without any further optimizations, train from sequences of a total length up to a million in minutes.

However, it is important to notice that the problem of generating test data has been chosen carefully as our first exercise, as it is far less computationally complex than using the model for prediction of genes or other structures. We discuss consequences of this observation in the concluding section.

4 Computational problems considered in the project

A major problem in using PRISM "as is" for large sequences is storage consumption. First of all the sequences themselves which, when represented as Prolog lists, take up far too much space and, parodically, in the application described above we needed to upgrade to a 64 bit architecture, which means the data size doubles so that one letter in a sequence occupies 24 bytes as opposed to 1 byte or even down to 2 bits in a traditionally programmed architecture for sequence analysis. Secondly, PRISM uses an explanation graph which may grow to the size of the input data or worse; we would need to identify when a model or part thereof can do with simpler learning principles, e.g., simple counting in

some cases. There exist techniques for compact representation of trees which may inspire to similar methods for graphs.

An automatic analysis may identify when a model resembles, say a HMM, so that the system can switch to a specialized implementation. The project group includes developers for competitive HMM based software [9] so we have the possibility to interface with existing software.

Execution time may also become a problem since PRISM calculates all probabilities as correct as possible, which means that it will trace all possible explanations for a given observation, including those with such a small probability that it does not contribute in any significant way. The project group includes expertise in automatic program analysis and semantics preserving transformation of logic programs. However, for programs with a probabilistic semantics and a very high complexity, some sort of pruning is necessary. Thus, new techniques are in demand which produce not necessarily correct results, but results which are guaranteed not to deviate less than some epsilon from the true results.

One possible way to reduce complexity seems to be by splitting the sequence. Analysis of a sequence typically takes time which is a steep function of the length of the sequence. Splitting it into pieces analyzed separately and combining results is a way of reducing time consumption. Analysis of a model may indicate where "good" split points may be identified and what is the loss of precision.

Pre-processing and additional annotation of data. Models are often trained from previously marked-up sequences, and pre-processing the sequences by adding additional annotations can reduce complexity of the training phase. As an example, in the application described in section 3 and described in further detail in [6], we applied a best-match algorithm to produce unique descriptions of the mutation errors in occurrences of repeat strings. This technique made the training time roughly proportional to the sequence length. Such preprocessing mechanisms should be studied further and generalized.

5 Proposals for enhanced expressibility

We have identified until now two dimensions where it can be relevant with additional expressibility compared to PRISM.

The first one concerns numerical distributions. Using a random variable for each letter to determine whether to continue generating or stop, results in a geometric distribution of (sub-) sequence lengths, and this principle is inherent in Markov based models as well as our example in section 2. However, a geometric distribution gives a relatively high probability to short sequences which in many cases is not appropriate to characterize the lengths of substrings of a given kind. For the genomic sequence model described in section 3 we coped with this problem for GC islands by assuming a fixed minimum length followed by a tail generated by a geometric distribution; this is a hack also often applied with HMMs. This is a very inelegant solution and in most cases it does not give a reliable model. We consider extending with facilities so that normal distributions

or perhaps a sort of "generic smooth distribution" can be described and learned from observational data.

The second topic concerns constraints and is more difficult. A stochastic sequence model in PRISM (or an HMM or SCFG) typically describes a distribution of what is possible, but have difficulties in capturing that some of these strings are not desired. Consider, for example, the model for intergenic subsequences described in section 3; here the random choices may also capture patterns that represent genes by accident, so to speak. This may happen more frequently that one would expect since the repeat substrings that occur may contain (mutated and fragmented) remains of obsolete genes.

It can obviously be a problem for an application that generates data. Intuitively, a possible solution might be to analyze the sequences produced with a new model that captures undesired patterns so that they can be discarded. However, the theoretical justification of this approach may be problematic as it indicates subtle dependencies between the random variables of the model.

6 Conclusion and discussion

In this paper, we have presented a research project whose intention is to study how biological sequence analysis may be improved using advanced logic-statistic modeling and machine learning techniques based on logic programming. This involves both getting more experience in using and developing such models as well as improving the modeling facilities andtheir implementation.

At time of writing, the project is just about starting up, so we have only few and early results to report, obtained from an experiment in generating artificial test data for gene finding in eukariotic genomic sequences. There is a biological focus on prokaryotic genetics in the project, which means that sequences are measured in thousands as opposed to millions, so we may hope to get biologically interesting results even if not all computational problems that we identified, have been solved.

As we noticed in section 3, the task of generating artificial sequences seems computationally far less complex than the task of prediction, which is to identify the most probable structure of an observed sequence, For this reason, the models applied for gene prediction tend to tune down the level of sophistication; most gene finders are based on HMMs. The general lower complexity of data generation (including the preceding learning phase) means that it is appropriate in general to use much more fine-grained and hopefully more reliable models for this part. This may suggest a methodology for development of gene finders, starting from (a) a detailed model, as can be developed in PRISM or other tools of similar expressibility, and from it develop (b) a model biased towards efficient implementation. During this process (a) can be applied continually to test (b) and get an indication of how much precision is sacrificed. We may also hope that our project may lead to a reduction of the distance between models of the (a) and (b) types.

Acknowledgement: This work is supported by the project "Logic-statistic modelling and analysis of biological sequence data" funded by the NABIIT program under the Danish Strategic Research Council, and the CONTROL project, funded by Danish Natural Science Research Council.

References

1. Sato, T., Kameya, Y.: Statistical abduction with tabulation. In Kakas, A.C., Sadri, F., eds.: Computational Logic: Logic Programming and Beyond. Volume 2408 of Lecture Notes in Computer Science., Springer (2002) 567–587
2. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. J. Artif. Intell. Res. (JAIR) **15** (2001) 391–454
3. Hobbs, J.R., Stickel, M.E., Appelt, D.E., Martin, P.A.: Interpretation as abduction. Artif. Intell. **63**(1-2) (1993) 69–142
4. Christiansen, H., Dahl, V.: Meaning in Context. In Dey, A., Kokinov, B., Leake, D., Turner, R., eds.: Proceedings of Fifth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-05). Volume 3554 of Lecture Notes in Artificial Intelligence. (2005) 97–111
5. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: ICLP. (1995) 715–729
6. Christiansen, H., Dahmcke, C.M.: A machine learning approach to test data generation: A case study in evaluation of gene finders. In: International Conference on Machine Learning and Data Mining MLDM'2007, Leipzig/Germany. Lecture Notes in Artificial Intelligence, Springer (2007) To appear.
7. Burset, M., Guigó, R.: Evaluation of Gene Structure Prediction Programs. Genomics **34**(3) (1996) 353–367
8. Guigó, R., Agarwal, P., Abril, J.F., Burset, M., Fickett, J.W.: An Assessment of Gene Prediction Accuracy in Large DNA Sequences. Genome Res. **10**(10) (2000) 1631–1642
9. http://www.clcbio.com/.

Reconsideration of Circumscriptive Induction with Pointwise Circumscription *

Koji Iwanuma[1], Katsumi Inoue[2], and Hidetomo Nabeshima[1]

[1] **University of Yamanashi**
4-3-11 Takeda, Kofu-shi,Yamanashi 400-8511, Japan
{iwanuma,nabesima}@iw.media.yamanashi.ac.jp
[2] **National Institute of Informatics**
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ki@nii.ac.jp

Abstract. Explanatory induction and descriptive induction are main frameworks for induction in logic. The both frameworks, however, have own some serious drawbacks; the explanatory induction often causes inductive leap problem and descriptive induction sometimes fail to explain given observations. Circumscriptive induction is a new framework intended to overcome these difficulties by unifying explanatory induction and descriptive induction. In this paper, we study and improve some aspects of circumscriptive induction. First, we enlarge the concept of inductive leaps. Secondly we clarify the sufficient condition for conservativeness with respect to generalised induction leaps, which shows that every correct solution for circumscriptive induction must be strongly conservative. Moreover we propose a new tractable induction framework, called pointwise circumscriptive induction. The pointwise induction just uses first-order logic with equality in the formulation, and does not demand any second-order computation. Pointwise induction enables us to derive some interesting hypotheses by the ordinary resolution performed in a mechanical way.

1 Introduction

There are two major frameworks for induction in logic, that is, *explanatory induction* and *descriptive induction* . Explanatory induction [18, 12] is the inference intended to induce general rules, called hypotheses, which explain the given observation under background knowledge. Descriptive induction [4, 12] induces hypotheses describing regularities confirmed by observations. Descriptive induction is increasing its own importance in the context of *data mining.* These two traditional frameworks have strength and weakness which are complementary to each other. For example, one of the most serious drawbacks in explanatory induction is inductive leaps, which deduce new facts not stated in given observations. Descriptive induction, on the other hand, often fails to induce hypotheses which can explain given observations.

* This research was partially supported by the Grant-in-Aid from The Ministry of Education, Science and Culture of Japan ((B)(1) No.17300051)

Inoue and Saito [6] proposed a novel induction framework, called *circumscriptive induction*, for integrating explanatory induction and descriptive induction. Circumscriptive induction is intended to possess two important properties: *completeness* for explaining given observations and *conservativeness* for causing no induction leaps.

In this paper, we study circumscriptive induction and its several aspects. Firstly we extend and reformulate the concept of inductive leaps, Secondly we simplify the sufficient condition for conservativeness relative to the generalized induction leaps. Moreover we show that a *correct solution* [6] to circumscriptive induction never involves induction leaps, which means that every correct solution must be conservative.

Circumscription, however, uses a second-order formulation, so is highly intractable for computing. As one approach for this computation problem, we propose a new tractable framework, called *pointwise circumscriptive induction.* Pointwise circumscription proposed by Lifschitz [13, 15] is a first-order approximation of circumscription. Thus pointwise circumscription induction never demands any second-order computation and often generates interesting hypothesis just with the ordinary resolutions.

This paper is organised as follows: Section 2 is preliminaries. In Section 3, we improve some aspects of circumscriptive induction. Section 4 is dedicated to pointwise circumscription and induction. Section 5 is for conclusion and future works.

2 Preliminaries

In this paper, we consider a *second-order language* $\mathcal{L}$ with the first-order equality [13]. We suppose $\mathcal{L}$ has predicate variables, but no function variables. We use a λ-notation for convenience to denote predicate expressions. An n-ary *predicate expression* is an expression of the form $\lambda\bar{x}.A(\bar{x})$, where $\bar{x}$ is a tuple of n individual variables $x_1, \ldots, x_n$and $A(\bar{x})$ is a formula possibly involving free occurrences of variables from $\bar{x}$. If α is an n-ary predicate expression $\lambda\bar{x}.A(\bar{x})$ and $\bar{t}$ is a tuple of n terms $t_1, \ldots, t_n$, then the expression $\alpha(\bar{t})$ stands for the formula $A(\bar{t})$. We identify a predicate constant P with the predicate expression $\lambda\bar{x}.P(\bar{x})$, and similarly for predicate variables.

Definition 1. Let $P_1, \ldots, P_n$ be predicate constants, and $\alpha_1, \ldots, \alpha_n$ be predicate expressions such that P_i and α_i are of the same arity ($1 \leq i \leq n$). $A[P_1/\alpha_1, \ldots, P_n/\alpha_n]$ is the formula obtained from a formula A by simultaneously replacing each occurrence $P_i(t)$ in A by the term $\alpha_i(t)$ (for each $i = 1, \ldots, n$).

A *clause* is a disjunction of literals

$$\neg A_1 \vee \cdots \vee \neg A_m \vee A_{m+1} \vee \cdots \vee A_n$$

where each A_i is an atom. Any variable in a clause is assumed to be universally quantified at the front. A clause is also written as:

$$A_1 \wedge \cdots \wedge A_m \supset A_{m+1} \vee \cdots \vee A_n$$

A *clausal theory* is a finite set of clauses. A clausal theory is identified with the conjunction of the clauses in it, and is also simply called a *formula*.

Let B, E and H be all clausal theories, where B, E and H are called a *background knowledge*, an *observation* and a *hypothesis*, respectively.

Definition 2 (explanatory induction [18, 12]). Given background knowledge B and an observation E, the task of *explanatory induction* is to infer a hypothesis H such that

$$B \wedge H \models E,$$

where $B \wedge H$ is consistent.

Definition 3 (descriptive induction [4, 12]). Given background knowledge B and an observation E, the task of *descriptive induction* is to infer a hypothesis H such that

$$Comp(B \wedge E) \models H,$$

where $Comp(B \wedge E)$ denotes the predicate completion of $B \wedge E$ relative to all predicates.

Example 1 (Inoue and Saito [6]). Let us consider the following background knowledge B_1 and the observation E_1:

$$B_1: \quad Bird(a) \wedge Bird(b);$$
$$E_1: \quad Flies(a).$$

The explanatory induction can generate the following hypothesis H_1:

$$H_1: \ \forall x(Bird(x) \supset Flies(x)).$$

The hypothesis H_1 satisfies $B_1 \wedge H_1 \models E_1$, but causes an *inductive leap*, that is, $B_1 \wedge H_1 \models Flies(b)$. On the other hand, descriptive induction produces the hypothesis H_2:

$$H_2 : \forall x(Flies(x) \supset Bird(x)).$$

The hypothesis H_2 never realizes an induction leap, i.e., $B_1 \wedge H_2 \not\models Flies(b)$. But H_2 fails to deduce E_1, i.e., $B_1 \wedge H_2 \not\models E_1$. These clearly show that explanatory induction and descriptive induction have their own strength and weakness which are complementary to each other.

Inoue [6] studied these difficulties, and proposed a new approach, called *circumscriptive induction* to overcome these shortages.

2.1 Circumscriptive Induction

Firstly we give the definition of circumscription in second-order language as follows: For any two predicate constants P and Q with the same arity, $(P \leq Q)$ is an abbreviation for $\forall x(P(\bar{x}) \supset Q(\bar{x}))$, where $\bar{x}$ is a tuple of distinct variables. Suppose $\bar{P} = P_1, \ldots, P_n$ and $\bar{Q} = Q_1 \ldots, Q_n$ are tuples of predicate constants such that P_i and Q_i have the same arity for $i = 1, \ldots, n$. Then, $\bar{P} \leq \bar{Q}$ stands for $P_1 \leq Q_1 \wedge \cdots \wedge P_n \leq Q_n$, and $\bar{P} < \bar{Q}$ stands for $(\bar{P} \leq \bar{Q}) \wedge \neg(\bar{Q} \leq \bar{P})$.

Definition 4 (parallel circumscription[13, 16]). Let $A(\bar{P}, \bar{Z})$ be a formula containing tuples $\bar{P}$ and $\bar{Z}$ of predicate constants $P_1, \ldots, P_m$ and $Z_1, \ldots, Z_n$, respectively. Suppose that p_i and z_j are predicate variables of the same arity as P_i and Z_j, respectively, for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. If we denote the tuple $p_1, \ldots, p_m$ by $\bar{p}$ and $z_1, \ldots z_n$ by $\bar{z}$, the *circumscription* $\mathrm{CIRC}[A; \bar{P}; \bar{Z}]$ *of* $\bar{P}$ *with* $\bar{Z}$ *varied* is defined as:

$$A(\bar{P}, \bar{Z}) \wedge \neg \exists pz(A[\bar{P}/\bar{p}, \bar{Z}/\bar{z}] \wedge (\bar{p} < \bar{P})).$$

Predicates $P_1, \ldots, P_m$ are called *minimised predicates*, and $Z_1, \ldots, Z_n$ are called *variable predicates*. When $\bar{Z}$ is empty, the circumscription is denoted as $\mathrm{CIRC}[A; \bar{P}]$.

In this paper, we consider only *standard models* for the equality. A *structure* M consists of a non-empty set $|M|$, called the *domain of individuals* , functions from $|M|^n$ to$|M|$ representing n-ary function constants, and subsets of $|M|^n$ representing n-array predicate constants. We write $|\mathcal{K}|_M$ to denote the extension of a (function or predicate) constant $\mathcal{K}$ in a structure M. The extension of a predicate expression $\lambda \bar{x}.A(\bar{x})$ in a structure M is also denoted as $|\lambda \bar{x}.A(\bar{x})|_M$. The equality $=$ is interpreted as the identity relation on $|M|$. A structure M is a *model* of a formula A, denoted as $M \models A$, if M satisfies A. Moreover, $\models A$ denotes that $M \models A$ for every structure M.

Definition 5. Let A be a formula, and $\bar{P}$ and $\bar{Z}$ be tuples of predicates $P_1, \ldots, P_m$ and $Z_1, \ldots, Z_n$, respectively. For any two models M and N of A, we write $M \leq^{\bar{P};\bar{Z}} N$ if:

1. $|M| = |N|$;
2. $|P_i|_M \subseteq |P_i|_N$ for every P_i in $\bar{P}$;
3. $|\mathcal{K}|_{\mathcal{M}} = |\mathcal{K}|_{\mathcal{N}}$ for every predicate/function constant $\mathcal{K}$ not in $\bar{P}$ nor in $\bar{Z}$.

The relation $\leq^{\bar{P};\bar{Z}}$ is a partial order. The expression $M =^{\bar{P};\bar{Z}} N$ denotes the case where both $M \leq^{\bar{P};\bar{Z}} N$ and $N \leq^{\bar{P};\bar{Z}} M$ hold. A model M of A is $\leq^{\bar{P};\bar{Z}}$*-minimal* if there is no model N such that $N \leq^{\bar{P};\bar{Z}} M$ and $M \neq^{\bar{P};\bar{Z}} N$.

For any formula B, we have [13, 16]

$\mathrm{CIRC}[A; \bar{P}; \bar{Z}] \models B$ if and only if B is satisfied by every $\leq^{\bar{P};\bar{Z}}$-minimal models of A.

Obviously, $\mathrm{CIRC}[A; \bar{P}; Z] \models \bigwedge_{P_i \in \bar{P}} \mathrm{CIRC}[A; P_i]$. The following is a well-known property:

Proposition 1 (well-foundedness [3]). *Any universal formula A is well-founded with respect to the order $\leq^{\bar{P};\bar{Z}}$, i.e., there is a $\leq^{\bar{P};\bar{Z}}$-minimal model for any universal formula A.*

Definition 6 (circumscriptive induction problem and correct solution [6]). Let B and E be clausal theories, and $\bar{P}$ and $\bar{Z}$ be disjoint tuples of predicates appearing in B and E. Then, a *circumscriptive induction problem* is defined as a quadruple $\langle B, E, \bar{P}, \bar{Z} \rangle$, where B and E represent background knowledge and an

observation, respectively. A formula H is a *correct solution to the circumscriptive induction problem* $\langle B, E, \bar{P}, \bar{Z}\rangle$ if

$$\mathrm{CIRC}[B \wedge E; \bar{P}; \bar{Z}] \models H \tag{1}$$

and

$$B \wedge H \models E, \tag{2}$$

where $B \wedge H$ is consistent.

Example 2 (continue). Reconsider the background knowledge B_1 and the observation E_1 in Example 1. Inoue and Saito [6] showed that circumscriptive induction allows us to generate the following hypothesis:

$$H_3 : \ \forall x(Bird(x) \wedge x \neq b \supset Flies(x)).$$

This hypothesis is a correct solution, that is, H_3 can explain E.

Remark 1. In the above example, if an additional fact $Bird(c)$ is added to the background knowledge B_1, then we can also deduce $Flies(c)$ with the hypothesis H_3. So the hypothesis H_3 is surely *inductive.*

Definition 7 (inductive leap, conservativeness and completeness [6]). Let A be a formula and P be a predicate constant. The *test set of inductive leaps* $TS(A, P)$ relative to A and P is

$$TS(A, P) = \{C \mid A \models C, \text{and } C \text{ is a ground atom whose predicate is } P\}.$$

Given a background B, an observation E and a hypothesis H, the hypothesis H *realizes an inductive leap* if there is a predicate P occurring positively in E such that

$$TS(B \wedge H, P) \not\subseteq TS(B \wedge E, P).$$

Otherwise, a hypothesis H is said to be *conservative*. A hypothesis H is said to be *complete* if for any predicate occurring positively in E, the following holds:

$$TS(B \wedge E, P) \subseteq TS(B \wedge H, P).$$

Note that the hypothesis H_3 in Example 2 is conservative and complete, whereas the hypothesis H_1 generated by explanatory induction in Example 1 is complete but not conservative. Moreover H_2 generated by descriptive induction is conservative but not complete.

Inoue and Saito [6] gave the following theorem of conservativeness just for *solitary formulas*.

Definition 8 (solitary formulas [13]). A formula $A(\bar{Z})$ is *solitary in* a tuple $\bar{Z}$ of predicates if $A(\bar{Z})$ can be written in the form of

$$Neg(\bar{Z}) \wedge (\bar{K} \leq \bar{Z}),$$

where $Neg(\bar{Z})$ is a formula not containing any predicate in $\bar{Z}$ positively, and $\bar{K}$ is a tuple of predicates not containing any predicate in $\bar{Z}$.

Variable predicates $\bar{Z}$ in a solitary formula $A(\bar{Z})$ can be eliminated in circumscription. That is, for any solitary formula $A(\bar{Z}) = \mathit{Neg}(\bar{Z}) \wedge (\bar{K} \leq \bar{Z})$, the following holds [13]:

$$\models \mathrm{CIRC}[A(\bar{Z}); \bar{P}; \bar{Z}] \equiv A(\bar{Z}) \wedge \mathrm{CIRC}[\mathit{Neg}(K); \bar{P}].$$

Theorem 1 (Conservativeness for solitary formulas [6]). *Let B and E be formulas such that $B \wedge E$ is solitary in predicates $\bar{Z}$. Suppose that H is a correct solution to the circumscriptive induction problem $\langle B, E, \bar{P}, \bar{Z}\rangle$. Then, H is conservative, i.e., H does not realize an inductive leap.*

3 Strong Conservativeness for Circumscriptive Induction

In this section we strengthen Theorem 1 in three aspects: the first is to enlarge the underlying vocabulary set of inductive leaps; the second is simplification of the sufficient condition for conservativeness of a circumscriptive hypothesis. This simplification leads to the guarantee of conservativeness for every correct solutions.

Definition 9 (general inductive leaps, strong conservativeness and completeness). Let $\bar{P}$ and $\bar{Z}$ be disjoint tuples of predicate constants. We define $\mathcal{L}(\bar{P}^{-}; \bar{Z})$ as the set of all formulas including no negative occurrences of any predicate in $\bar{P}$ nor no occurrences of any predicate in $\bar{Z}$. Suppose A is a formula. The *general test set of inductive leaps* $\mathit{GTS}(A; \bar{P}; \bar{Z})$ relative to A, $\bar{P}$ and $\bar{Z}$ is

$$\mathit{GTS}(A; \bar{P}; \bar{Z}) = \{\ C \in \mathcal{L}(\bar{P}^{-}; \bar{Z}) \mid A \models C\ \}.$$

Let $\langle B, E, \bar{P}, \bar{Z}\rangle$ be a circumscriptive induction problem and H be a formula. If

$$\mathit{GTS}(B \wedge H; \bar{P}; \bar{Z}) \not\subseteq \mathit{GTS}(B \wedge E; \bar{P}; \bar{Z}),$$

then we say, a hypothesis H *realizes a general inductive leap*. Otherwise, a hypothesis H is said to be *strongly conservative*. A hypothesis H is said to be *strongly complete* if for any predicate occurring in E, the following holds:

$$\mathit{GTS}(B \wedge E, P) \subseteq \mathit{GTS}(B \wedge H, P).$$

Compared with $TS(A, \bar{P}, \bar{Z})$, the general test set $\mathit{GTS}(A; \bar{P}; \bar{Z})$ may contain various forms of formulas, For example, the following disjunctive formulas are belong to the underlying set $\mathcal{L}(\bar{P}^{-}, \bar{Z})$:

$$\begin{aligned} D_1 :&\quad P_1(t_1) \vee \cdots \vee P_1(t_k), \\ D_2 :&\quad P_1(s) \vee P_2(s), \end{aligned}$$

where the terms $t_1, \ldots, t_k$ and s may possibly contain variables. Thus, if D_1 and D_2 are logical consequences of A, then $D_1, D_2 \in \mathit{GTS}(A; \bar{P}; \bar{Z})$. These non-ground disjunctive formulas can not be treated by $TS(A; \bar{P}; \bar{Z})$ at all.

Next we clarify the sufficient condition for strong conservativeness.

Theorem 2 (A sufficient condition for strong conservativeness). *Let* $\langle B, E, \bar{P}, \bar{Z}\rangle$ *be a circumscriptive induction problem, and* H *be a formula. If*

$$\mathrm{CIRC}[B \wedge E; \bar{P}; \bar{Z}] \models H,$$

then H *is strongly conservative, i.e.,*

$$GTS(B \wedge H; \bar{P}; \bar{Z}) \subseteq GTS(B \wedge E; \bar{P}; \bar{Z}).$$

Proof. Let C be an arbitrary formula belonging to $GTS(B \wedge H; \bar{P}; \bar{Z})$. From the assumption $\mathrm{CIRC}[B \wedge E; \bar{P}; \bar{Z}] \models H$, the formula $B \wedge H$ is true on all $\leq^{\bar{P};\bar{Z}}$-minimal model of $B \wedge E$. Therefore, C is also true on any $\leq^{\bar{P};\bar{Z}}$-minimal model of $B \wedge E$. Since $B \wedge E$ is a well-founded theory, every model M of $B \wedge E$ has a $\leq^{\bar{P};\bar{Z}}$-minimal model N such that $N \leq^{\bar{P};\bar{Z}} M$. For these models M and N of $B \wedge E$, we have

1. C is true on N;
2. $|P_i|_N \subseteq |P_i|_M$ for any $P_i \in \bar{P}$;
3. $|Q|_N = |Q|_M$ for any predicate Q not belonging to $\bar{P}$ nor $\bar{Z}$.

The formula C consists of only positive occurrences of predicates in $\bar{P}$ and occurrences of predicates Q not in $\bar{P}$ nor $\bar{Z}$. Therefore, it is obvious that C is true on M. Thus $C \in GTS(B \wedge E; \bar{P}; \bar{Z})$. □

Note that if a hypothesis H is strongly conservative, the H is also conservative in the sense of Definition 7. The following is the first main theorem.

Theorem 3 (Strong conservativeness of correct solutions). *Let* $\langle B, E, \bar{P}, \bar{Z}\rangle$ *be a circumscriptive induction problem. Any correct solution* H *of* $\langle B, E, \bar{P}, \bar{Z}\rangle$ *is strongly conservative and complete for* B *and* E*.*

Proof. According to the definition 6 and 9, any correct solution H is obviously complete to B and H. Conservativeness is an immediate consequence of Theorem 3. □

Note that Theorem 3 holds for any given background B and any observation E and any correct solution H. Hence Theorem 4 is really a proper extension of Theorem 1.

4 Pointwise Circumscription: An Approximation Method

Circumscription is a very powerful tool, but is rather intractable for computing because it uses a second-order formulation. Many researches have been conducted on mechanical computation of circumscription [2, 8–11, 13, 14, 16, 20]. Inoue [6] investigated the computation problem of circumscriptive induction, and proposed two mechanical methods: the first one is to apply some equivalent transformation methods of circumscription into first-order formulas [2, 8, 10, 11,

16] and successively apply first-order theorem provers [5] to the resulting first-order formulas; the second one is to directly use some explanatory induction provers [7, 18] and apply native circumscriptive theorem provers [10, 20] successively. In this paper, we propose a new third method, i.e., an *approximation method based on pointwise circumscription*.

Pointwise circumscription was proposed by Lifschitz [14, 15] as a first-order approximation of parallel circumscription *without variable predicates*. Although many extended versions were proposed [10], we present here the most basic form of pointwise circumscription [13] for simplicity for discussion.

Let $\bar{s}$ and $\bar{t}$ be tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ of n terms, respectively. We write $\bar{s} = \bar{t}$ to denote the conjunction $\bigwedge_{i=1}^{n}(s_i = t_i)$. The expression $\bar{s} \neq \bar{t}$ stands for $\neg(\bar{s} = \bar{t})$.

Definition 10 (pointwise circumscription [14, 15]). Let A be a sentence, P be an n-ary predicate constant and $\bar{x}$ be a tuple of n individual variables not appearing in A. The *pointwise circumscription* $\mathrm{PWC}[A; P]$ of P in A is the first-order sentence

$$A \wedge \forall \bar{x}\Big[P(\bar{x}) \supset \neg\Big(A[P/\lambda \bar{u}(P(\bar{u}) \wedge \bar{u} \neq \bar{x})]\Big)\Big].$$

The subformula $\neg(A[P/\lambda u(P(\bar{u}) \wedge \bar{u} \neq \bar{x})])$ in $\mathrm{PWC}[A; P]$ is called the *pointwise definition formula*, denoted as $\mathrm{Pwf}[A; P; \bar{x}]$.

We have clearly

$$\models \mathrm{CIRC}[A; P] \supset \mathrm{PWC}[A; P].$$

The formula $\mathrm{PWC}[A; P]$ semantically states that it is impossible to obtain a model of A by eliminating *exactly one element* from the extension of P. The formula $\mathrm{PWC}[A; P]$ behaves as an elementary approximation [3] of $\mathrm{CIRC}[A; P]$. Moreover $\mathrm{PWC}[A; P]$ logically implies some extended forms of predicate completion. [4]

Although pointwise circumscription can not treat with variable predicates variables directly, several variable predicates $\bar{Z}$ in $\mathrm{CIRC}[A; \bar{P}; \bar{Z}]$ can be eliminated. That is, $\mathrm{CIRC}[A; \bar{P}; \bar{Z}]$ can be equivalently transformed into $\mathrm{CIRC}[A'; \bar{P}]$ sometimes. In Appendix, we will give some new methods for eliminating variable predicates from parallel circumscription over a *positive/negative solitary formula relative to variable predicates*. Moreover we have, for any $\mathrm{CIRC}[A; \bar{P}; \bar{Z}]$,

$$\models \mathrm{CIRC}[A; \bar{P}; \bar{Z}] \supset \mathrm{CIRC}[A; \bar{P}].$$

Thus, together with $\models \mathrm{CIRC}[A; P] \supset \mathrm{PWC}[A; P]$, we have generally

$$\models \mathrm{CIRC}[A; \bar{P}; \bar{Z}] \supset \mathrm{PWC}[A; \bar{P}].$$

[3] The formula $\mathrm{PWC}[A; P]$ can be regarded as a first-order approximation of circumscription. Even in the framework of first-order logic, more elaborate approximation formulas can be considered. Such approximation can be regarded as higher-oder approximation formulas. See [10] for the detail.

[4] See [17] for Horn theories, and [8] for arbitrary first-order theories.

Therefore pointwise circumscription can be regarded as a first approximation for parallel circumscription even if it involves variable predicates.

Therefore, we next define a new induction framework and a correct solution etc. with pointwise circumscription.

Definition 11 (pointwise induction problem). Let B and E be formulas, and $\bar{P}$ be a tuple of predicates appearing in B and E. Then, a *pointwise induction problem* is defined as the triple $\langle B, E, \bar{P}\rangle$, where B and E represent background knowledge and an observation, respectively.

Definition 12 (correct solution and conservativeness). Given an induction problem $\langle B, E, \bar{P}\rangle$, a formula H is a *correct solution to the pointwise induction problem* if

$$\bigwedge_{P_i \in \bar{P}} \mathrm{PWC}[B \wedge E; \bar{P}_i] \models H \tag{3}$$

and

$$B \wedge H \models E, \tag{4}$$

where $B \wedge H$ is consistent. Moreover, we say, a hypothesis H *realizes an general inductive leap* if

$$GTS(B \wedge H; \bar{P}; \phi) \not\subseteq GTS(B \wedge E; \bar{P}; \phi).$$

Otherwise, a hypothesis H is said to be *strongly conservative*.

Theorem 4. *Let $\langle B, E, \bar{P}\rangle$ be a pointwise induction problem, and H be a formula. If*

$$\bigwedge_{P_i \in \bar{P}} \mathrm{PWC}[B \wedge E; P_i] \models H,$$

then H is strongly conservative, i.e.,

$$GTS(B \wedge H; \bar{P}; \phi) \subseteq GTS(B \wedge E; \bar{P}; \phi).$$

Proof. This theorem can be proved in a quite similar way to Theorem 2. We shall omit the proof here. □

The following corollary clarifies that pointwise induction can be regarded as an approximation of circumscriptive induction.

Corollary 1. *Let B and E be formulas. Suppose that H is a correct solution to the pointwise induction problem $\langle B, E, \bar{P}\rangle$. Then, H is a correct solution to the circumscriptive induction problem $\langle B, E, \bar{P}, \phi\rangle$.*

Proof. Since $\models \mathrm{CIRC}[B \wedge E; \bar{P}] \supset \bigwedge_{P_i \in \bar{P}} \mathrm{PWC}[B \wedge E; \bar{P}]$, this is obvious. □

Remark 2. For $P_i \in \bar{P}$, since $\mathrm{CIRC}[B \wedge E; \bar{B}; \bar{Z}] \models \mathrm{PWC}[B \wedge E; P_i]$ and $\mathrm{PWC}[B \wedge E; P_i] \models E$ hold, $PWC[B \wedge E; P_i]$ is always guaranteed to be a correct solution to circumscriptive induction problem $\langle B, E, \bar{B}, \bar{Z}\rangle$. The formula $\mathrm{PWC}[B \wedge E; P_i]$, however, contains E as its subformula (see Definition 10), so it should be a meaningless as a hypothesis, or at least, not an interesting solution.

Next, we consider a conservative and/or correct hypothesis H for pointwise circumscription. At first, we clarify a relationship between Clark's predicate completion [1] and $\mathrm{Pwf}[A; P; x]$.

Definition 13 (minimal extension formulas [8]). Let A be a conjunction $A_1 \wedge \ldots \wedge A_m$ of formulas, P be an n-ary predicate constant and $\bar{x}$ be a tuple of n individual variables not appearing in A. The *minimal extension formula* $\mathrm{Min}[A; P; \bar{x}]$ of P in A is the formula $\neg B$, where B is obtained from A by replacing every positive occurrence of P in A as follows:

1. if an occurrence $P(\bar{t})$ is in a definite clause A_i, then we replace the occurrence $P(\bar{t})$ by $\bar{x} \neq \bar{t}$.
2. Otherwise, we replace the occurrence $P(\bar{t})$ with $P(\bar{t}) \wedge (\bar{x} \neq \bar{t})$.

The following lemma is valuable for simplifying the minimal extension formula.

Lemma 1 (Iwanuma et. al.[8]). *Let A be a conjunction $A_1 \wedge \ldots \wedge A_n$, $\bar{P}$ be a tuple of predicate constants $P_1, \ldots, P_m$. $\mathrm{Sim}_{\bar{P}}(A)$ is defined as the conjunction obtained from A by eliminating any conjuncts A_i not containing any positive occurrences of a predicate in $\bar{P}$. We have*

1. $\models \mathrm{CIRC}[A; \bar{P}] \equiv A \wedge \mathrm{CIRC}[\mathrm{Sim}_{\bar{P}}(A); \bar{P}]$;
2. $\models \mathrm{PWC}[A; P_i] \equiv A \wedge \mathrm{PWC}[\mathrm{Sim}_{\{P_i\}}(A); P_i]$.

Example 3. Let B_2 and E_2 be formulas as follows:

$$B_2 : \quad Bird(a) \wedge Bird(b) \wedge Bird(c)$$
$$E_2 : \quad Flies(a) \wedge Flies(b)$$

The formula $\mathrm{Min}[\mathrm{Sim}_{Bird}(B_2 \wedge E_2); Bird; x]$ is as follows:

$$x = a \vee x = b \vee x = c.$$

The formula $\mathrm{Min}[\mathrm{Sim}_{Flies}(B_2 \wedge E_2); Flies; x]$ is

$$x = a \vee x = b.$$

Example 4. Let B_3 be the disjunctive formula:

$$Bird(a) \vee Bird(b).$$

Suppose E_2 is the same one as in Example 3. The formula $\mathrm{Min}[\mathrm{Sim}_{Bird}(B_3 \wedge E_2); Bird; x]$ is

$$(Bird(a) \supset x = a) \wedge (Bird(b) \supset x = b).$$

Lemma 2 ([8]). *Let A be a first-order formula and P be a predicate constant. We have*

1. $A \models \forall \bar{x}(\mathrm{Min}[A; P; \bar{x}] \supset P(\bar{x}))$;

2. $\models \forall \bar{x}(\mathrm{Pwf}[A;P;\bar{x}] \supset \mathrm{Min}[A;P;\bar{x}])$.

Therefore, we have

1. $\models \mathrm{PWC}[A;P] \supset \forall \bar{x}(P(\bar{x}) \equiv \mathrm{Pwf}[A;P;\bar{x}])$;
2. $\models \mathrm{PWC}[A;P] \supset \forall \bar{x}(P(\bar{x}) \equiv \mathrm{Min}[A;P;\bar{x}])$.

The minimal extension formula may often be valuable to generate a conservative/complete hypothesis for pointwise induction. Let us consider two examples.

Example 5. Let B_2 and E_2 be formulas in Example 3. Consider the minimal extension formulas with respect to $\mathrm{PWC}[A;Bird;x]$ and $\mathrm{PWC}[A;Flies;x]$. The formula

$$\forall x(Bird(x) \equiv \mathrm{Min}[\mathrm{Sim}_{Bird}(B_2 \wedge E_2);Bird;x])$$

is the conjunction of the following clauses:

$$\begin{aligned} C1: &\quad x = a \supset Bird(x).\\ C2: &\quad x = b \supset Bird(x).\\ C3: &\quad x = c \supset Bird(x).\\ C4: &\quad Bird(x) \supset (x = a \vee x = b \vee x = c). \end{aligned}$$

On the other hand, the equivalence formula

$$\forall x(Flies(x) \equiv \mathrm{Min}[\mathrm{Sim}_{Flies}(B_2 \wedge E_2);Flies;x])$$

is the conjunction of the following clauses:

$$\begin{aligned} C5: &\quad x = a \supset Flies(x).\\ C6: &\quad x = b \supset Flies(x).\\ C7: &\quad Flies(x) \supset (x = a \vee x = b). \end{aligned}$$

All of the clauses shown above are logical consequences from $\mathrm{PWC}[\mathrm{Sim}_{Bird}(B_2 \wedge E_2);Bird;x]$ and $\mathrm{PWC}[\mathrm{Sim}_{Flies}(B_2 \wedge E_2);Flies;x]$. We resolve the clause $C4$ with the clauses $C5$ and $C6$ successively, and obtain the resolvent $(Bird(x) \supset Flies(x) \vee x = c)$, which is equivalent to

$$C8: \ Bird(x) \wedge x \neq c \supset Flies(x).$$

This formula is not only a correct solution to the pointwise circumscriptive induction problem $\langle B_2, E_2, \{Bird, Flies\}\rangle$, but also is a correct solution to the circumscriptive induction problem $\langle B_2, E_2, \{Bird, Flies\}; \phi\rangle$.

Remark 3. The clauses $C5$ and $C6$ are logically equivalent to the formula E_2 itself. Thus the clause $C8$ is indeed a correct solution to the circumscriptive (or pointwise circumscriptive) induction problem $\langle B_2, E_2, \{Bird\}, \phi\rangle$. Another important fact is that $C8$ can be derived only by performing the ordinary resolution, where no special computation mechanism is needed. Inoue and Saito [6] derived a similar formula from circumscriptive induction, *by hand and by using some heuristics.* Such a derivation task is not so simple nor easy. This shows the superiority of pointwise circumscription with respect to tractability.

Example 6. Let B_3 be a disjunctive formula in Example 3 and E_3 be the unit clause

$$E_3 : \quad Flies(a).$$

The formula $\forall(Bird(x) \equiv \mathrm{Min}[\mathrm{Sim}_{Bird}(B_3 \wedge E_3); Bird; x])$ is the conjunction of the following clauses:

$$\begin{aligned} C9 : &\quad [(Bird(a) \supset x = a) \wedge (Bird(b) \supset x = b)] \supset Bird(x). \\ C10 : &\quad Bird(x) \wedge Bird(a) \supset x = a. \\ C11 : &\quad Bird(x) \wedge Bird(b) \supset x = b. \end{aligned}$$

All of these clauses are logical consequences from $\mathrm{PWC}[\mathrm{Sim}_{Bird}(B_3 \wedge E_3); Bird; x]$. On the other hand, the observation E_3 is equivalent to the clause $C5$. Thus, we resolve the clause $C10$ with the clause $C5$, and obtain the resolvent

$$C12 : \ Bird(x) \wedge Bird(a) \supset Flies(x).$$

Clearly we have $\mathrm{PWC}[B_3 \wedge E_3; Bird] \models C12$, but unfortunately, $B_3 \wedge C12 \not\models E_3$. That is, the hypothesis $C12$ is strongly conservative, but is not complete. The formula $C12$ should be regarded as a *conditional solution*, because $C12$ refers the case when the predicate $Bird$ entails $Flies(a)$, that is, the case when $Bird(a)$ is true. Therefore, *abduction* seems to be an appropriate method for continuing further inference computation. In other words, circumscriptive induction plays a role of clarifying which part of knowledge should be handled by abduction.

5 Conclusion

In this paper, we studied circumscriptive induction and elaborated its some aspects. We extended the concept of inductive leaps, and give a simple sufficient condition for conservativeness with respect to the general induction leaps. Moreover we propose a new tractable framework, pointwise circumscriptive induction instead of circumscriptive one. The pointwise induction needs first-order logic with equality, and does not demand any second-order computation, so is rather tractable for computing.

One of the most interesting future-work is to extend pointwise circumscription framework to a more sophisticated one such as extended pointwise circumscription given in [10]. Another interesting work is a conditional solution, which seems to be closely relative to abductive computation, and is inevitable for treating disjunctive background theories and observations.

References

1. Keith L. Clark: Negation as failure. In: H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pp.119–140, Plenum Press (1978)
2. P. Doherty, W. Łukaszewicz and A. Szałas: Computing circumscription revisited: Preliminary report, in: *Proc. of Inter. Joint Conf. on Artificial Intelligence '95* 1502–1507 (1995)

3. D. Etherington: *Reasoning with Incomplete Information* (Pitman, London, 1988).
4. Nicolas Heft: Induction as nonmonotonic inference. *Proc. of KR'89.* pp.149–156 (1989)
5. Katsumi Inoue: Linear resolution for consequence finding, *Artificial Intelligence* , 56, 301–353 (1992)
6. Katsumi Inoue and Haruka Saito: Circumscription Polices for Induction. *Proc. of the 14th Intern. Conf. on Inductive Logic Programming (ILP), Lecture Notes in Artificial Intelligence.* 3194, pp.164–179 (2004)
7. Katsumi Inoue: Induction as consequence finding. *Machine Learning*, 55, pp.109–135 (2004).
8. Koji Iwanuma, Masateru Harao and Shoichi Noguchi: Reconsideration of Pointwise Circumscription. *Proc. of Inter. Conf. on Information Technology Harmonising with Society (InfoJapan'90)* pp.147–154 (1990)
9. Koji Iwanuma: Conservative query normalization on parallel circumscription, in: *Proc. of 12th Inter. Conf. on Automated Deduction (CADE-12), Lecture Notes in Artificial Intelligence* 814 296-310 (1994)
10. Koji Iwanuma, Kazuhiko Oota: An extension of pointwise circumscription. *Artificial Intelligence* 86, pp.291–402 (1996)
11. P.G. Kolaitis and C.H. Papadimitriou: Some computational aspects of circumscription, *Journal of the ACM* 37(1), 1–14 (1990)
12. Nicolas Lachiche: Abduction and induction form a non-monotonic reasoning perspective. In: P.A. Flach and A. Kakas, editors, *Abduction and Induction: Essay on their Relation and Integration*, Kluwer Academic (2000)
13. Vladimir Lifschitz: Computing circumscription. *Proc. of the 9th Inter. Joint Conf. on Artificial Intelligence,* pp.122-127 (1985)
14. Vladimir Lifschitz: Pointwise circumscription: preliminary report, in: *Proc. of AAAI-86*, 406-410 (1986)
15. Vladimir Lifschitz: Pointwise circumscription, in: M.L. Ginsberg editors., *Readings in Nonmonotonic Reasoning* 179-193 (Morgan Kaufman Pub., 1988)
16. Vladimir Lifschitz: Circumscription. In: D.M. Gabbay C.J. Hogger and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol.3, pp.298–352, Oxford University Press (1994)
17. Y. Moinard and R. Rolland, Circumscription and definability, in: *Proc, of Inter. Joint Conf. on Artificial Intelligence '91*, 432-437 (1991)
18. Stephen Muggleton: Inverse entailment and Progol. *New Generation Computing.* 13, pp.245–286 (1995)
19. Hans. J. Ohlbach: SCAN—elimination of predicate quantifies. *Proc. of the 13th Inter. Conf. on Automated Deduction (CADE'1996) Lecture Notes in Artificial Intelligence* , 1104, pp.161–165 (1996)
20. T.C. Przymusinski, An algorithm to compute circumscription, *Artificial. Intelligence.* **38** (1989) 49-73.

Appendix: Variable Predicate Elimination Method for Positive/Negative Solitary Formulas

We show a new method for eliminating variable predicates from parallel circumscription over extended solitary formulas. The following is a key property.

Lemma 3. *Let $A(\bar{Z})$ be a formula containing a tuple $\bar{Z}$ of predicate constants $Z_1, \ldots, Z_n$. Suppose that W_i is a predicate expression of the same arity as Z_i for $i = 1, \ldots, n$, and we denote the tuple $W_1, \ldots W_n$ by $\bar{W}$. If*

$$\models A(\bar{Z}) \supset A[\bar{Z}/\bar{W}],$$

then we have, for any tuple $\bar{P}$ of predicate constants,

$$\models \ \mathrm{CIRC}[A(\bar{Z}); \bar{P}; \bar{Z}] \ \equiv \ A \wedge \mathrm{CIRC}[A[\bar{Z}/\bar{W}]; \bar{P}; \bar{Z}].$$

Proof. Firstly we prove the "if" part. Suppose M is a $\leq^{\bar{P};\bar{Z}}$-minimal model of $A[\bar{Z}/\bar{W}]$ and M is also a model of $A(\bar{Z})$. Then M must be $\leq^{\bar{P};\bar{Z}}$-minimal among all models of $A(\bar{Z})$. If not so, there is a model N of $A(\bar{Z})$ such that $N \leq^{\bar{P};\bar{Z}} M$ and $N \neq^{\bar{P};\bar{Z}} M$. Since $\models A(\bar{Z}) \supset A[\bar{Z}/\bar{W}]$, the model N is also a model of $A[\bar{Z}/\bar{W}]$. This contradicts the $\leq^{\bar{P};\bar{Z}}$-minimality of M among the models of $A[\bar{Z}/\bar{W}]$.

Next we prove the "only if" part. Suppose M is a $\leq^{\bar{P};\bar{Z}}$-minimal model of $A(\bar{Z})$. Then M is also a model of $A[\bar{Z}/\bar{W}]$. If M is not $\leq^{\bar{P};\bar{Z}}$-minimal among the models of $A[\bar{Z}/\bar{W}]$, then there exits a model N of $A[\bar{Z}/\bar{W}]$ such that $N \leq^{\bar{P};\bar{Z}} M$ and $N \neq^{\bar{P};\bar{Z}} M$. For these models M and N, we have

1. $|N| = |M|$;
2. $|P_i|_N \subset |P_i|_M$ for every $P_i \in \bar{P}$;
3. $|\mathcal{K}|_N = |\mathcal{K}|_M$ for every function/predicate constant $\mathcal{K}$ not in $\bar{P}$ nor in $\bar{Z}$.

We construct a new structure K from N such that

1. $|K| = |N|$;
2. $|Z_i|_K = |W_i|_N$ for every Z_i in $\bar{Z}$;
3. $|\mathcal{K}|_K = |\mathcal{K}|_N$ for all function/predicate constants other than $Z_1, \ldots, Z_n$.

The structure K should be a model of $A(\bar{Z})$, because N is a model of $A[\bar{Z}/\bar{W}]$. Moreover, the following hold:

1. $|K| = |M|$;
2. $|P_i|_K \subset |P_i|_M$ for every $P_i \in \bar{P}$;
3. $|\mathcal{K}|_N = |\mathcal{K}|_M$ for every function/predicate constant $\mathcal{K}$ not in $\bar{P}$ nor in $\bar{Z}$.

This contradicts the minimality of M among the models of $A(\bar{Z})$. □

If $A[\bar{Z}/\bar{W}]$ has no occurrences of predicates in $\bar{Z}$, then $\mathrm{CIRC}[A[\bar{Z}/\bar{W}]; \bar{P}; \bar{Z}]$ can obviously be reduced into $\mathrm{CIRC}[A[\bar{Z}/\bar{W}]; \bar{P}]$ by eliminating variable predicates $\bar{Z}$. In this case, $\mathrm{CIRC}[A(\bar{Z}); \bar{P}; \bar{Z}]$ itself can eventually be transformed to $A \wedge \mathrm{CIRC}[A[\bar{Z}/\bar{W}]; \bar{P}]$.

Definition 14 (positive/negative solitary formulas). We call the solitary formula given in Definition 8 an *positive solitary formula*. A *negative solitary formula* is the dual of a positive solitary formula, that is, a formula in the form of

$$\mathit{Pos}(\bar{Z}) \wedge (\bar{Z} \leq \bar{K}),$$

where $\mathit{Pos}(\bar{Z})$ is a formula not containing any predicate in $\bar{Z}$ negatively, and $\bar{K}$ is a tuple of predicates not containing any predicate in $\bar{P}$.

Obviously, we have

$$\begin{aligned} &\models \quad [Neg(\bar{Z}) \wedge (\bar{K} \leq \bar{Z})] \supset [Neg(\bar{K}) \wedge (\bar{K} \leq \bar{K})]. \\ &\models \quad [Pos(\bar{Z}) \wedge (\bar{Z} \leq \bar{K})] \supset [Pos(\bar{K}) \wedge (\bar{K} \leq \bar{K})]. \end{aligned}$$

Therefore, according to Lemma 3, the following elimination of variable predicates from parallel circumscription is available. Notice that the first one is well-known result given by Lifschitz [13].

Corollary 2 (Variable elimination from positive/negative solitary formulas). *Let A be a formula, and $\bar{P}$ and $\bar{Z}$ be disjoint tuples of predicate constants. We have:*

1. *(Lifschitz [13]) If A is a positive solitary formula $Neg(\bar{Z}) \wedge (\bar{K} \leq \bar{Z})$, then we have*
$$\models \mathrm{CIRC}[A; \bar{P}; \bar{Z}] \equiv A \wedge \mathrm{CIRC}[Neg(\bar{K}); \bar{P}];$$
2. *If A is a negative solitary formula $Pos(\bar{Z}) \wedge (\bar{Z} \leq \bar{K})$, then we have*
$$\models \mathrm{CIRC}[A; \bar{P}; \bar{Z}] \equiv A \wedge \mathrm{CIRC}[Pos(\bar{K}); \bar{P}].$$

Nonmonotonic Abductive Inductive Learning

Oliver Ray[1]

University of Bristol
United Kingdom
oray@cs.bris.ac.uk

Abstract. Inductive Logic Programming (ILP) is concerned with the generalization of examples with respect to prior knowledge expressed in a logic program formalism. Negation as Failure (NAF) is a key logic programming feature that can enable the representation and learning of defaults and exceptions. But most ILP research has been aimed at the special case of Horn programs and has so far failed to exploit the full potential of NAF. By contrast, Abductive Logic Programming (ALP), a closely related task concerned with explaining observations relative to a background theory, has been extensively studied and applied in the context of normal clauses. This paper shows how ALP can be used to lift practical ILP methods from Horn theories to normal logic programs. In particular, it formalises an approach called eXtended Hybrid Abductive Inductive Learning (XHAIL) that integrates ALP and ILP in a common reasoning framework driven by language and search bias. The utility of XHAIL is shown by a nonmonotonic Event Calculus (EC) case study where NAF is used to model the persistence of properties through time.

1 Introduction

Inductive Logic Programming (ILP) [22] is a branch of Artificial Intelligence (AI) concerned with the generalization of positive and negative examples with respect to prior background knowledge expressed in a logic program formalism. Compared to other AI representations, logic programs are expressive and easy to understand. Moreover, Negation as Failure (NAF) [3] gives logic programming a nonmonotonic inference mechanism for reasoning with defaults and exceptions under incomplete information. Since incompleteness is an inherent feature of any learning problem, effective utilization of NAF is potentially a major strength of the ILP paradigm.

To date, most ILP research has been aimed at pure Horn clause programs without NAF. Some approaches have been proposed for normal programs with NAF such as [2, 10, 4, 14, 16, 7, 30, 8] and more recently in [29, 23]. But these can only be used in rather restricted cases or they lack efficient strategies for guiding the computation. Because many of the pruning techniques successfully used in the Horn clause setting are not applicable in the presence of NAF, it is essential for nonmonotonic ILP to make full use of available language and search bias. But, while effective mechanisms have been developed in the Horn case, these have yet to be satisfactorily applied in the general case.

This work aims to develop a practical nonmonotonic ILP method by lifting successful methods of language and search bias from Horn clauses to normal programs. The proposal exploits techniques from a closely related branch of AI, called Abductive Logic Programming (ALP) [12], which grew from attempts to provide a semantics and proof procedure for NAF [5]. Simple forms of abductive reasoning have already been used in ILP for learning rules with predicates not mentioned in the examples [21, 18]. Here, the idea is to also exploit the relation between ALP and NAF to enable the learning of normal logic programs.

This paper formalises the method of eXtended Hybrid Abductive Inductive Learning (XHAIL) which lifts the Hybrid Abductive Inductive Learning (HAIL) approach of [27] from Horn clauses to normal programs. The technique is based on the construction and generalization of a preliminary ground hypothesis called a *Kernel Set* [26] that serves to bound the search space in accordance with user specified bias. The result is a three stage process whereby abduction and deduction are used compute the head and body literals of a Kernel Set, which is then generalised by an inductive subsumption-based search procedure.

The rest of the paper is structured as follows. Section 2 introduces the key notation and background information. Section 3 describes the XHAIL approach and shows how all three phases can be implemented using an ALP interpreter to handle NAF while exploiting language and search bias. Section 4 illustrates XHAIL on a case study involving a nonmonotonic Event Calculus (EC) [13] framework for reasoning about action and change. This paper extends the earlier abstract in [25] by providing a more detailed description of XHAIL, further discussion of related work, and a larger biologically motivated case study.

2 Background

This section introduces the key notation and terminology used in the paper. Section 2.1 reviews some basic definitions of logic programs [15], defines the stable model semantics [9], and outlines the task of ALP [12]. Section 2.2 recalls the language bias mechanism of mode declarations [20] and describes the task of ILP [22]. Section 2.3 summarises the HAIL approach [27] for Horn clauses.

2.1 Abductive Logic Programming (ALP)

This paper assumes a first order language (without equality) containing at least one constant and function symbol. An atom is a predicate p (of arity n) followed by an n-tuple of terms $(t_1, \ldots, t_n)$. A literal is an atom a (a positive literal) or its negation *not* a (a negative literal). A (normal) clause is an expression of the form $a \leftarrow l_1, \ldots, l_m$ where a is an atom (called the head atom) and the l_i are literals (called body literals). A constraint is a clause of the form $\bot \leftarrow l_1, \ldots, l_m$ where $\bot$ is an atom denoting logical falsity. A (logic) program is a set of clauses and constraints. A program is Horn iff all of its literals are positive and is normal otherwise. A (Herbrand) interpretation is a set of ground atoms. An interpretation I satisfies a positive ground literal $l = a$ iff $a \in I$ and

it satisfies a negative ground literal $l = not\ a$ iff $a \notin I$. It satisfies a set of ground literals iff it satisfies each ground atom in the set. It satisfies a ground clause iff it satisfies the head atom or fails to satisfy at least one body literal.

A (Herbrand) model M of a program P is an interpretation I that satisfies every ground instance of every clause C in P. A model M is minimal if no strict subset is a model of P. Moreover, it is stable if M is the unique minimal model of the Horn program P^M obtained from the ground instances of P by removing all clauses with a negative literal not satisfied in M and removing all negative literals from the remaining clauses. A program P satisfies a set of ground atoms G if a stable model of P satisfies G. The satisfaction of G in a stable model of P is denoted $P \models G$. [1] A clause C is said to (θ-)subsume a clause D iff there is a substitution θ such that the head atom of D is the head atom of $C\theta$ and each body literal in D is contained in $C\theta$. A program P is said to (θ-)subsume a program Q iff every clause in Q is subsumed by a clause in P.

ALP seeks to compute the conditions under which a goal G can be made to succeed from a theory T. In this paper it suffices to consider the case when G is a set of literals, T is a set of clauses, and the task of ALP is to compute a set of ground atoms Δ, called an explanation, together with a ground substitution θ, called an answer substitution, such that $T \cup \Delta \models G\theta$. The atoms in Δ are usually restricted to a set A of predicates called abducibles, which identify predicates for which only partial information is available in the theory (e.g., potential faults in a diagnosis task or possible actions in a planning domain). In this case, the pair (Δ, θ) is called an abductive solution of G wrt. T and A. Hereafter, the notation $ALP(T, G, A, r)$ will represent the set of all abductive solutions of G wrt. T and A that are computable within r resolution steps.

An explanation Δ is minimal if no strict subset of Δ is an explanation. Many ALP systems can compute such explanations, but this paper assumes an efficient variant of the prominent Kakas-Mancarella (KM) procedure [12] called ProLogICA [28]. This exploits the basic KM approach of by interleaving abductive and consistency computations, but includes several pruning techniques that greatly improve the efficiency of the procedure.

2.2 Inductive Logic Programming (ILP)

ILP seeks to compute a set of hypotheses H that explain a set of (positive and negative) examples E with respect to a background theory B. In this paper it suffices to consider the case when E is a set of ground (positive and negative) literals, and B and H are programs. Given B and E as inputs, the task of ILP is to compute a set of clauses H such that $B \cup H \models E$. The clauses in H are usually constrained by some form of domain dependent language bias. This paper uses a well known form of language bias called mode declarations. As defined in [20] and briefly recalled below, a set M of mode declarations defines a language or hypothesis space $\mathcal{L}_M$ within which $H \subseteq \mathcal{L}_M$ must fall.

[1] Note that, if P is a consistent Horn Program, then $\models$ coincides with the classical first order entailment relation.

In brief, a mode declaration m is either a head declaration $modeh(r, l)$ or a body declaration $modeb(r, s)$ where r is an integer (the recall) and s is a ground literal (the scheme) possibly containing so-called placemarker terms of the form $\#type$, $+type$ and $-type$, where $type$ is a type predicate. Schemes can be seen as 'templates' with placemarkers standing for constants, input variables and output variables, respectively. The distinction between input and output variables is given by the restriction that any input variable in a body literal must be an input variable in the head or an output variable in some preceding body literal. This is consistent with standard logic programming usage whereby input (resp. output) variables are bound before (resp. after) a goal is called.

In this way, a set M of mode declarations is associated with a set of clauses $\mathcal{L}_M$, called the language of M, such that $C = a \leftarrow l_1, \ldots, l_n \in \mathcal{L}_M$ iff the head atom a (resp. each body literal l_i) is obtained from some head (resp. body) declaration in M by replacing all # placemarkers with constants and by replacing all + (resp. −) placemarkers with input (resp. output) variables. For any mode declaration m, $pred(m)$ denotes the predicate p at the front of the scheme s, $schema(m)$ denotes the literal obtained from s by replacing all placemarkers with distinct variables $X_1, \ldots, X_n$, and $type(m)$ denotes the sequence of literals $t_1(X_1), \ldots, t_n(X_n)$ such that t_i is the type predicate of the placemarker replaced by the variable X_i. For any set of mode declarations M, M^+ is the set of head declarations in M, and M^- is the set of body declarations in M.

As explained in [20], and used in Section 3, the mode declaration recall is used to bound the number of times the scheme is used, and the types are used to restrict the terms that replace a placemarker during hypothesis construction. The symbol $*$ is used to indicate an arbitrarily large recall and a default predicate any is assumed for any placemarker where the type is omitted. In addition to the satisfying language bias specified by a set of mode declarations, ILP hypotheses are often required to satisfy additional criteria. This paper uses a heuristic called compression which embodies the principle of Occam's Razor (which advocates choosing the simplest hypothesis that fits the data), in this case, by preferring hypotheses containing the fewest number of literals [20].

2.3 Hybrid Abductive Inductive Learning (HAIL)

HAIL is a mode-directed approach for finding compressive ILP hypotheses in the Horn case (when B and H are NAF free). Given a background theory B, an example or set of examples E, and mode declarations M, it computes a hypothesis $H \subseteq \mathcal{L}_M$ containing as few literals as possible such that $B \cup H \models E$. To solve this task, XHAIL uses the language bias M to construct and generalise a preliminary ground hypothesis K called a Kernel Set of B and E.

The idea is to construct H in three stages by returning three progressively more complex hypotheses In essence: stage 1 first produces a set of ground atoms Δ that collectively entail E when added to B; stage 2 then produces a set of ground clauses K whose head atoms are those in Δ and whose body atoms are entailed by B; and finally stage 3 produces a (consistent and compressive) hypothesis H that subsumes K.

Intuitively, the head atoms of K are a set of atoms that, if true, would entail E,. By contrast, the body atoms of K are atoms that are already known to be true. Thus, the Kernel Set K entails E with respect to B; and so does any theory H that subsumes K. Importantly, it is easier to invert subsumption than it is to invert entailment. The trick is to efficiently exploit the mode declarations M when computing Δ, K and H. For this, HAIL uses a multi-clause generalisation of the widely used Progol methodology [20, 21].

– In stage 1, HAIL computes a set of ground atoms

$$\Delta = \left\{ \begin{array}{l} \alpha_1 \\ \vdots \\ \alpha_n \end{array} \right\}$$

such that $B \cup \Delta \models E$ and each atom α_i in Δ is a well-typed ground instance of a clause in the language $\mathcal{L}_{M^+}$ of the head declarations M^+. These atoms are computed by an abductive procedure that returns an explanation Δ of the goal E wrt. the program B augmented with a set of type clauses extracted from the head declarations.

– In stage 2, HAIL computes a set of ground clauses

$$K = \left\{ \begin{array}{l} \alpha_1 \leftarrow \delta_1^1, \ldots, \delta_1^{m_1} \\ \vdots \\ \alpha_n \leftarrow \delta_n^1, \ldots, \delta_n^{m_n} \end{array} \right\}$$

such that $B \models \delta_i^j$ for all $1 \leq i \leq n, 1 \leq j \leq m_i$ and each clause $\alpha_i \leftarrow \delta_i^1, \ldots, \delta_1^{m_i}$ in K is a well-typed ground instance of a clause in $\mathcal{L}_M$. The body literals of each clause are computed by a deductive procedure that finds the successful ground instances of the queries obtained by substituting a set of input terms into the + placemarkers of the body declaration schemas.

– In stage 3, HAIL computes a set of clauses

$$H = \left\{ \begin{array}{l} a_1 \leftarrow d_1^1, \ldots, d_1^{m'_1} \\ \vdots \\ a_n \leftarrow d_n^1, \ldots, d_n^{m'_n} \end{array} \right\}$$

such that $B \cup H \models E$ and each clause $a_i \leftarrow d_i^1, \ldots, d_i^{m'_i}$ in H is in $\mathcal{L}_M$ and subsumes the corresponding clause $\alpha_i \leftarrow \delta_i^1, \ldots, \delta_1^{m_i}$ in K. This is done by (greedily) generalising each Kernel Set clause, in turn, using a top down A* like search of the subsumption lattice to find a maximally compressive clause that is in the language of M and is consistent with the other clauses.

3 eXtended Hybrid Abductive Inductive Learning (XHAIL)

This section introduces XHAIL, which lifts the HAIL methodology from Horn clauses to normal programs. Like its predecessor, the underlying idea of XHAIL is to use language and search bias to guide the computation by constructing and generalising a preliminary ground Kernel Set. The key issue is how to generalise the abductive, deductive, and inductive phases to handle NAF.

The approach described below is based on transforming each phase into a separate ALP task. The advantages of this approach are that (i) implementation is simplified by having a single primary reasoning engine, (ii) all three phases benefit from efficiency enhancements to the underlying engine, (iii) other NAF semantics can easily be supported by simply using a different ALP system.

The XHAIL algorithm is formalised in Figure 1 below. The inputs consist of a program B (background theory), a set of ground literals E (positive and negative examples) a set mode declarations M (language bias), and two integers d and r (depth and resolution bounds). The output is a compressive program H (hypothesis) such that $B \cup H \models E$ and $H \in \mathcal{L}_M$.

Like HAIL, XHAIL computes the hypothesis in three phases. Each phase contains one or more statements (each marked with an asterix in Figure 1) which are choice points where backtracking may be necessary to find an optimal solution. Each of the three phases are described in more detail below and the key differences with respect to the Horn case are discussed.

3.1 Abductive Phase

As in the Horn case, the first stage of XHAIL must compute a set of ground unit program $\Delta = \bigcup_{i=1}^{n} \alpha_i$ such that $B \cup \Delta \models E$ and each atom α_i is a well-typed ground instance of a clause in $\mathcal{L}_{M^+}$. The only difference is that a full ALP system must be used that supports NAF. Because each abduced atom will go in the head of a Kernel Set clause, the abducibles A_1 are obtained from the set of predicates appearing p/n at the front of some head declaration scheme. But, to ensure any abduced atoms satisfy the language bias, for each such predicate, a clause is created of the form $p(X_1, \ldots, X_n) \leftarrow p^*(X_1, \ldots, X_n), p'(X_1, \ldots, X_n)$ containing two fresh predicates p^*/n and p'/n. In effect, p^* identifies the ground instances of a that satisfy the head declarations. This is done by adding one clause of the form $schema^*(m) \leftarrow type(m)$ for each head declaration $m \in M^+$. [2] The other predicate p' is just an abducible proxy for the original predicate p. As soon as a set X of abducibles is computed, any occurrence of p' is replaced by p to give a well-formed hypothesis Δ. One hypothesis must be chosen before proceeding to the next stage. The number of choices is greatly reduced by only considering minimal explanations. This is also a useful heuristic as it results in Kernel Sets with the fewest number of clauses — which are easier to construct and generalise and are likely to result in compressive hypotheses.

[2] An almost identical set of typing clauses is used by Progol.

INPUTS:

program B (background theory), ground literals E (examples),
mode declarations M (bias), integers d and r (depth bounds)

% abductive phase %

let A_1 be the set of predicates containing one fresh predicate p'/n for each predicate $p/n = pred(m)$ at the front of the scheme of some head declaration $m \in M^+$

let T_1 be the set of clauses obtained by adding to B one clause $p(X_1, \ldots, X_n) \leftarrow p'(X_1, \ldots, X_n), p^*(X_1, \ldots, X_n)$ for each predicate $p'/n \in A_1$ and one clause $p^*(X_1, \ldots, X_n) \leftarrow t_1(X_1), \ldots, t_n(X_n)$ for each head declaration $m \in M^+$ with schema $p(X_1, \ldots, X_n)$ and types $t_1(X_1), \ldots, t_n(X_n)$

*__let__ X be any explanation in $ALP(T_1, E, A_1, r)$

let Δ be the set of ground atoms obtained by replacing each atom $p'(t_1, \ldots, t_n) \in X$ with the corresponding atom $p(t_1, \ldots, t_n)$

% deductive phase %

let A_2 be the empty set $\emptyset$

let T_2 be the set of clauses B

for each atom $\alpha_i \in \Delta$

*__let__ m_i be any head declaration in M whose schema subsumes α_i

set n_i to the set of terms in α_i corresponding to + placemarkers in m_i

set k_i to the unit clause $\alpha_i \leftarrow$

repeat up to d times

let Q be the set of goals of the form $type(m), schema(m)\phi$ where $m \in M^-$ is a body declaration and ϕ is a substitution binding all variables in $schema(m)$ that replaced a + placemarker to a term in n_i

let R be the set of ground literals of the form $schema(m)\phi\theta$ where $schema(m)\phi$ appears in a goal $G \in Q$ such that the computation $ALP(T_2, G, A_2, r)$ succeeds with the answer substitution θ (and an empty set of assumptions)

*__let__ S be any subset of R

add to the body of k_i all literals in S (not already in k_i)

add to n_i all (new) terms in S corresponding to − place-markers in M

let k' be the set of clauses obtained from k by replacing all distinct terms corresponding to + and − placemarkers with fresh variables

let K be the set of clauses $\{k_1, \ldots, k_n\}$

let K' be the set of clauses $\{k'_1, \ldots, k'_n\}$

% inductive phase %

let A_3 be the set of predicates $\{use\}$

let T_3 be the set of clauses obtained by adding to B two clauses $try(I, J, C) \leftarrow not\ use(I, J)$ and $try(I, J, C) \leftarrow use(I, J), C$ along with one clause of the form $\alpha'_i \leftarrow try(i, 1, {\delta'}_i^1), \ldots, try(i, m_i, {\delta'}_i^{m_i})$ for each $k_i = \alpha'_i \leftarrow {\delta'}_i^1, \ldots, {\delta'}_i^{m_i} \in K'$

*__let__ Y be any explanation in $ALP(T_3, E, A_3, r)$

let H be the set of clauses obtained from K by removing every body atom δ_i^j for which the corresponding abducible $use(i, j)$ is not contained in Y

OUTPUT:

program H (hypothesis)

Fig. 1. XHAIL proof procedure (*showing choice points)

3.2 Deductive Phase

As in the Horn case, the second stage of XHAIL must compute a ground program $K = \bigcup_{i=1}^{n} \alpha_i \leftarrow \delta_i^1, \ldots, \delta_i^{m_i}$ where $B \models \delta_i^j$ for all $1 \leq i \leq n, 1 \leq j \leq m_i$ and each clause $\alpha_i \leftarrow \delta_i^1, \ldots, \delta_1^{m_i}$ is a well-typed ground instance of a clause in $\mathcal{L}_M$. To do this, each head atom is then taken, in turn, and saturated with body literals by means of a nonmonotonic generalisation of the level saturation procedure used by Progol [20]. For this purpose, the ALP system is made to behave as a deductive query answering procedure by declaring an empty set $\emptyset$ of abducibles (so that only negative literals may be assumed). [3] Each head atom is then processed by choosing a head declaration m_i to initialise a growing reservoir of input terms n_i which are substituted into the + placemarkers of any body declarations m to generate a set of goals Q whose successful ground instances, $ALP(B, G, \emptyset, r)$, result in a set of literals R which can be added into the body of the clause k_i with head atom α_i. Some subset S of R must then be added to k_i and any output terms added to n_i. The clause k_i resulting from each abducible α_i is then added to the Kernel Set K.

It is interesting to note that, unlike the the Horn case, it does not necessarily follow that $B \cup K \models E$. For this to hold, all of the body literals in K would need to be true in the *same* model of B and they would also have to be true in *some* model of $B \cup \Delta$. [4] To ensure the second condition is satisfied, the ALP interpreter was modified to recognise a set of declarations of the form $consistent(\alpha_i)$ for each atom $\alpha_i \in \Delta$. Not allowing a consistency computation to succeed by assuming the negation of these atoms means that the body atoms in K are all true in a stable model of B and remain true in some stable model of $B \cup \Delta$. To ensure the first condition is satisfied, it is sufficient to accumulate the assumptions made as each body atom is computed. In this way, the head atoms will be supported by a set of body literals that are true before and after K is added to B. In Example 1 below, K_1 satisfies this condition and is a correct hypothesis, but K_2 and K_3 violate this condition and are not.

Enforcing these two conditions reduces the number and size of candidate Kernel Sets and thereby makes the generalisation task easier. Relaxing them has the opposite effect, and is simply a way of weakening the Kernel Set bias.

Example 1. Let

$$B = \left\{ \begin{array}{l} a \leftarrow not\ e \\ b \leftarrow e \\ c \end{array} \right\} \quad \text{and} \quad E = \{\, e \,\} = \Delta$$

and

$$K_1 = \{\, e \leftarrow c \,\} \quad \text{and} \quad K_2 = \{\, e \leftarrow b \,\} \quad \text{and} \quad K_3 = \{\, e \leftarrow a \,\}$$

[3] As noted in [5] the ALP treatment of NAF is often superior to SLDNF used in most Prolog systems as the former performs caching on negative literals which can avoid repeated computations and avoid infinite loops through negative recursion.

[4] In the Horn case, the first condition is guaranteed because there is only one stable model and the second condition is satisfied because of monotonicity. In the non-monotonic case, both of these may no longer be true.

3.3 Inductive Phase

As in the Horn case, the third stage of XHAIL computes a program $H = \bigcup_{i=1}^{n} a_i \leftarrow d_i^1, \ldots, d_i^{m_i'}$ such that $B \cup H \models E$ and each clause $a_i \leftarrow d_i^1, \ldots, d_i^{m_i'}$ is in $\mathcal{L}_M$ and subsumes the corresponding clause $\alpha_i \leftarrow \delta_i^1, \ldots, \delta_i^{m_i}$ in K. Strictly, the hypothesis H should not subsume the Kernel Set K, but should subsume the set of clauses K' obtained from K by replacing all input and output terms with variables. This essentially amounts to deleting as many body literals from K' as possible. Two syntactic transformations prepare the ALP system for this task through the introduction of two new predicates $try/3$ and $use/2$. First, two clauses $try(I, J, C) \leftarrow not\ use(I, J)$ and $try(I, J, C) \leftarrow use(I, J), C$ are created. Then, each body literal $\delta_i'^j$ in K' is wrapped inside a meta-predicate of the form $try(i, j, \delta_i'^j)$. Applying an ALP procedure to the resulting theory, with the goal E and one abducible predicate use, returns a set Y of ground atoms of the form $use(i, j)$, which indicate that the corresponding literals $\delta_i'^j$ should be included in H and the others should not.

In other words, the literal $use(i, j)$ means use the jth literal in the ith clause of K'. The intuition underlying this approach is that in order to use a head atom α_i' from K' in some derivation of E, the ALP procedure must solve each of the body atoms $try(i, j, \delta_i'^j)$. By the two rules added to K', each such atom can be solved in one of two ways: either by assuming $not(use(i, j))$; or by abducing $use(i, j)$ and solving $\delta_i'^j$. The former effectively ignores $\delta_i'^j$ as if it were not there, while the latter solves $\delta_i'^j$ as if it were part of the clause. In this way, each explanation records which atoms from K' should be included in H and which should not. One such explanation must be chosen and used to determine the final hypothesis H. Minimal abductive explanations correspond 1-1 to maximally compressive inductive generalisations. If the correct linking of input and output variables is required, then additional constraints would be needed, as illustrated in Example 2 below, but these are omitted from Figure 1.

Example 2. If

$$K' = p(X) \leftarrow q(X, Y), q(Y, Z) \quad \text{and} \quad M = \left\{ \begin{array}{l} modeh(*, p(+)) \\ modeb(*, q(+, -)) \end{array} \right\}$$

then the clause

$$p(X) \leftarrow try(1, 1, q(X, Y)), try(1, 2, q(Y, Z))$$

is constructed along with the constraint

$$\bot \leftarrow use(1, 2), not\ use(1, 1)$$

stating we cannot use the 2nd atom if we do not also use the 1st atom.

Soundness of XHAIL follows from the soundness of the ALP procedure used in the inductive phase. Termination of XHAIL is ensured by the depth bounds d and r. Note that, in the non-monotonic case, (a) an incremental approach can *not* be used to generalise one seed example at a time and (b) Horn clause pruning techniques cannot be used in the inductive phase.

4 Learning Event Calculus (EC) Domain Theories

This section illustrates the ability of XHAIL to learn normal programs using a temporal model of lactose metabolism in the bacterium E. coli [11]. The model is based on a nonmonotonic EC [13] framework for reasoning about action and change. This paper employs the Simplified EC formalisation of [31] stating how the truth of certain fluents (properties) change as certain events (actions) occur. This framework is used to model how lactose metabolism is regulated by the presence of the sugars lactose and glucose in the E. coli growth medium. In brief, E. coli prefers to feed on glucose but can also feed on lactose by producing extra enzymes that essentially break down lactose into glucose. In order to conserve energy, the bacterium only produces these enzymes when lactose is available as a food source but glucose is not. The aim is to learn this mechanism from a narrative stating how the lactose metabolism is enabled and disabled in response to the addition and subtraction of lactose and glucose from the growth medium.

The inputs to XHAIL are shown in Figure 2. The theory B consists of four parts: the core EC axioms, the type predicates, a partial domain theory, and a narrative. The first 3 lines in B are the core EC axioms. They state that, a fluent F is true (*holdsAt*) at a time $T2$ if (i) an event E occurred (*happened*) at some earlier time $T1$ which caused F to become true (*initiated*) and no intervening event falsified (*clipped* or *terminated*) F, or (ii) F was originally (*initially*) true and was not clipped in the mean time. [5] The next 5 lines declare the *time*-points used in the model as well as the *event*s (whether lactose/glucose is added/subtracted) and the $fluent$s (whether lactose/glucose is present and whether lactose is being metabolised). The next 4 lines define a partial domain theory. They state that adding (resp. subtracting) lactose/glucose at any time T is sufficient to ensure the presence (resp. absence) of lactose/glucose. [6] The last 8 lines specify the fluents that initially hold, and the events which then occur.

The examples E state at which time points lactose metabolism was known to hold. The aim is to learn the initiating and terminating conditions for lactose metabolism. This amounts to completing the initial domain axioms. Hence, the mode declarations M state that hypothesized clauses may have atoms of the form $initiates(e, f, T)$ and $terminates(e, f, T)$ in their head, where e and f are constants (fluents and events) and T is a variable (a time); and literals of the form $holdsAt(f, T)$ and $not\ holdsAt(f, T)$ in their bodies where f is a fluent and T is a variable in the head. Note that, for simplicity, this example only uses one event per time point and it provides a complete scenario for the fluent $meta(lact)$. However, the XHAIL methodology also works for more complex examples involving concurrent actions and partial scenario. By adding head declarations for *initially* and/or *happens* the method can also be used for incomplete narratives.

Consider now the result of applying XHAIL to the inputs in Figure 2.

[5] Note that NAF is necessary to model the persistence of fluents through time i.e. a fluent is assumed to remain true if there is no evidence it was clipped.

[6] Note that by the axioms in Figure 1, the truth of any fluents initiated or terminated by an action at time T will only change at the next time point $T + 1$.

%— **Background Knowledge (B)**—%

holdsAt(F,T2) ← happens(E,T1), T1<T2, initiates(E,F,T1), not clipped(T1,F,T2).
holdsAt(F,T2) ← initially(F), not clipped(0,F,T2).
clipped(T1,F,T2) ← happens(E,T), T1<T, T<T2, terminates(E,F,T).

time(0). time(1). time(2). time(3). time(4).
time(5). time(6). time(7). time(8). time(9).
event(add(gluc)). event(add(lact)). event(sub(gluc)). event(sub(lact)).
fluent(pres(lact)). fluent(pres(gluc)). fluent(meta(lact)).

initiates(add(gluc),pres(gluc),T).
initiates(add(lact),pres(lact),T).
terminates(sub(gluc),pres(gluc),T).
terminates(sub(lact),pres(lact),T).

initially(pres(gluc)).
happens(add(lact),1).
happens(sub(gluc),2).
happens(sub(lact),3).
happens(add(lact),4).
happens(add(gluc),5).
happens(sub(lact),6).
happens(sub(gluc),7).

%— **Examples (E)**—%

not holdsAt(meta(lact),1),
not holdsAt(meta(lact),2),
holdsAt(meta(lact),3),
not holdsAt(meta(lact),4),
holdsAt(meta(lact),5),
not holdsAt(meta(lact),6),
not holdsAt(meta(lact),7),
not holdsAt(meta(lact),8).

%— **Mode Declarations (M)**—%

modeh(*,initiates(#event,#fluent,+time)).
modeh(*,terminates(#event,#fluent,+time)).
modeb(*,holdsAt(#fluent,+time)).
modeb(*,not holdsAt(#fluent,+time)).

Fig. 2. XHAIL Inputs

The abductive phase results in the addition of the type clauses

$$
\begin{array}{l}
initiates(A,B,C) \leftarrow initiates^*(A,B,C), initiates'(A,B,C).\\
terminates(A,B,C) \leftarrow terminates^*(A,B,C), terminates'(A,B,C).\\
initiates^*(A,B,C) \leftarrow event(A), fluent(B), time(C).\\
terminates^*(A,B,C) \leftarrow event(A), fluent(B), time(C).
\end{array}
$$

along with the abducibles

$$initiates' \quad \text{and} \quad terminates'$$

and returns exactly one minimal explanation

$$X = \left\{ \begin{array}{l} initiates'(sub(gluc), meta(lact), 2),\\ terminates'(sub(lact), meta(lact), 3),\\ initiates'(add(lact), meta(lact), 4),\\ terminates'(add(gluc), meta(lact), 5). \end{array} \right\}$$

which, after removing primes, results in the hypothesis

$$\Delta = \left\{ \begin{array}{l} initiates(sub(gluc), meta(lact), 2),\\ terminates(sub(lact), meta(lact), 3),\\ initiates(add(lact), meta(lact), 4),\\ terminates(add(gluc), meta(lact), 5). \end{array} \right\}$$

The deductive phase results in the addition of the declarations

$$
\begin{array}{l}
consistent(initiates(sub(gluc), meta(lact), 2)),\\
consistent(terminates(sub(lact), meta(lact), 3)),\\
consistent(initiates(add(lact), meta(lact), 4)),\\
consistent(terminates(add(gluc), meta(lact), 5)).
\end{array}
$$

so, saturating the first head atom $initiates(sub(gluc), meta(lact), 2)$, gives

$$n_i = \{2\}$$

$$Q = \left\{ \begin{array}{l} fluent(F), holdsAt(F, 2)\\ fluent(F), not\ holdsAt(F, 2) \end{array} \right\}$$

$$R = \left\{ \begin{array}{l} holdsAt(pres(lact), 2)\\ holdsAt(pres(gluc), 2)\\ not\ holdsAt(meta(lact), 2) \end{array} \right\}$$

whereupon, choosing $S = R$, produces

$$
\begin{array}{rl}
k_1 = initiates(sub(gluc), meta(lact), 2) \leftarrow & holdsAt(pres(lact), 2),\\
& holdsAt(pres(gluc), 2),\\
& not\ holdsAt(meta(lact), 2).
\end{array}
$$

which results in

$$k_1' = initiates(sub(gluc), meta(lact), X) \leftarrow \begin{array}{l} holdsAt(pres(lact), X), \\ holdsAt(pres(gluc), X), \\ not\ holdsAt(meta(lact), X). \end{array}$$

so that, saturating the remaining atoms in the same way, finally gives

$$K' = \left\{ \begin{array}{lll} initiates(sub(gluc), meta(lact), X) & \leftarrow & \begin{array}{l} holdsAt(pres(lact), X), \\ holdsAt(pres(gluc), X), \\ not\ holdsAt(meta(lact), X). \end{array} \\ terminates(sub(lact), meta(lact), X) & \leftarrow & \begin{array}{l} holdsAt(pres(lact), X), \\ not\ holdsAt(pres(gluc), X). \end{array} \\ initiates(add(lact), meta(lact), X) & \leftarrow & \begin{array}{l} not\ holdsAt(pres(lact), X), \\ not\ holdsAt(pres(gluc), X). \end{array} \\ terminates(add(gluc), meta(lact), X) & \leftarrow & \begin{array}{l} holdsAt(pres(lact), X), \\ not\ holdsAt(pres(gluc), X). \end{array} \end{array} \right\}$$

The inductive phase results in the addition of the clauses

$$\begin{array}{rcl} initiates(sub(gluc), meta(lact), X) & \leftarrow & \begin{array}{l} try(1, 1, holdsAt(pres(lact), X)), \\ try(1, 2, holdsAt(pres(gluc), X)), \\ try(1, 3, not\ holdsAt(meta(lact), X)). \end{array} \\ terminates(sub(lact), meta(lact), X) & \leftarrow & \begin{array}{l} try(2, 1, holdsAt(pres(lact), X)), \\ try(2, 2, not\ holdsAt(pres(gluc), X)). \end{array} \\ initiates(add(lact), meta(lact), X) & \leftarrow & \begin{array}{l} try(3, 1, not\ holdsAt(pres(lact), X)), \\ try(3, 2, not\ holdsAt(pres(gluc), X)). \end{array} \\ terminates(add(gluc), meta(lact), X) & \leftarrow & \begin{array}{l} try(4, 1, holdsAt(pres(lact), X)), \\ try(4, 2, not\ holdsAt(pres(gluc), X)). \end{array} \\ try(X, Y, Z) & \leftarrow & not\ use(X, Y). \\ try(X, Y, Z) & \leftarrow & use(X, Y), Z. \end{array}$$

along with the abducible

$$use$$

and returns exactly one minimal hypothesis

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

corresponding to the maximally compressive hypothesis

$$H = \left\{ \begin{array}{l} initiates(sub(gluc), meta(lact), T) \leftarrow holdsAt(pres(lact), T). \\ terminates(sub(lact), meta(lact), T). \\ initiates(add(lact), meta(lact), T) \leftarrow not\ holdsAt(pres(gluc), T). \\ terminates(add(gluc), meta(lact), T). \end{array} \right\}$$

This example illustrates several key features of XHAIL:

- it performs non-Observation Predicate Learning (non-OPL) [21] as none of the predicates defined in H are defined in E;
- it reasons through negations as the 4th literal in E yields the 2nd atom in Δ via a negated clipped literal;
- it uses language bias effectively as it bounds the search space using a set K of just 4 clauses containing only 13 literals in total;
- it performs true multiple predicate learning as all of the clauses in K are searched in parallel to produce an H defining two predicates.

Moreover, the XHAIL framework in Figure 1 is clearly parametric on the choice of ALP algorithm. This means that the ALP algorithm can be chosen to suit a particular application. In this paper it was assumed that the ALP system behaves according to the stable model semantics. However, it is equally possible to use an ALP procedure designed for any other preferred model semantics such as the perfect or well-founded models. In this sense the XHAIL framework is also parametric on the choice of logic program semantics. In fact, the XHAIL methodology has also been implemented using an answer set solver. Preliminary experiments suggest that an answer solver is considerably more efficient than ALP when applied to the inductive phase of the procedure. [7]

5 Related Work

Moyle [17] describes an application of Alecto to the learning of domain specific EC axioms similar to the case study above. Unlike XHAIL, Alecto requires several transformations to 'decouple' the learning of the *initiates* and *terminates* predicates by artificially generating integrity constraints from the observations to represent the fact that certain fluents are not *clipped* [18]. Unfortunately, we were unable to verify these results since the Alecto semantics (described in [18]) and the only publicly available proof procedure (included in [32]) apply to just Horn programs.

Moyle and Muggleton [19] describe an application of Progol5 to the same problem of learning of domain specific EC axioms. In addition to the decoupling transformations mentioned above, the limitations of Progol5's abductive reasoning procedure also necessitate a rewriting of the EC axioms to express the *initiates* and *terminates* predicates into one *flips* predicate with reified truth

[7] In order to apply the answer set solver, typing literals have to be added to all program clauses and the meta-predicate *try* must be rewritten using the technique in [25]

values. While this just leaves a single negated clipped atom in the theory, The inability of Progol5 to reason abductively through negation means that it can only solve a restricted class of problems (and, in particular, cannot solve the case study presented in this paper).

Otero [24] has shown that some temporal induction problems (in a variation of the Situation Calculus) can be reduced to monotonic reasoning by exploiting causal structure. Otero [23] considers the task of nonmonotonic induction under the skeptical stable model semantics. He gives necessary and sufficient conditions for the existence of a stable model solution and describes a method for computing the corresponding stable models. But, his method only returns grount unit hypotheses (i.e., complete sets of ground atoms) and does not utilise any form of language or search bias.

Sakama [29] proposes an nonmonotonic ILP methodology (for extended logic programs) based on the sceptical stable model semantics. However the method is restricted to OPL learning (where hypothesis and example predicates coincide) and it requires that the example predicates appear nowhere in the theory. As mentioned in [29], overcoming these restrictions would necessitate the use of abductive techniques, and this is exactly what XHAIL achieves. The procedure in [29] does use the notions of 'relevance' and 'involvement' to constrain the search space, but it does not exploit bias to the same extent as XHAIL. Unlike XHAIL, which generalizes all of the examples in one go, the procedure in [29] considers examples one by one which introduces a dependency on the order in which examples are presented to the system and forces additional restrictions on the interdependencies among example predicates. Also, [29] is restricted to learning rules that are 'negative cycle free'.

Esposito et al [6] describe a system INTHELEX which applies abductive and inductive operators to learn hierarchical datalog programs under the so-called 'object identity assumption'. In principle, INTHELEX can be used to perform non-monotonic induction. However, after rewriting our case study into a suitable form (to exclude function symbols and built-in predicates) we were unable to learn the required hypothesis using INTHELEX.

A survey of other related nonmonotonic ILP is provided in [29] that includes [2, 10, 4, 14, 16, 7, 30, 8] Although a detailed comparison remains to be carried out, it is clear that for certain problems XHAIL overcomes some of the practical limitations of these systems.

Alrajeh et al [1] have applied XHAIL to the learning of operational system requirements from scenarios. They use an EC formalism augmented to represent multiple scenarios and event preconditions.

6 Conclusions

This paper formalised the XHAIL framework for learning normal logic programs, and illustrated this approach on a biologically motivated case study using the EC formalism for reasoning about action and change. Since it is likely that such calculi will become more important in biological and other applications, effective

(nonmonotonic) techniques for inferring domain theories are much needed. Unlike other approaches for nonmonotonic ILP, XHAIL uses language and search bias to bound the search space by constructing and generalising a preliminary Kernel Set. It was shown how abductive techniques can be used to implement all three phases of the XHAIL approach. This shows that abductive reasoning can be usefully exploited in nonmonotonic induction procedures and it provides a methodology that may be of use in other learning problems. However, the limitations of XHAIL need to be studied more closely and it remains to validate the method on a more realistic case study.

Acknowledgments

I am grateful to Dalal Alrajeh, Krysia Broda, Antonis Kakas and Alessandra Russo for useful discussions. This work was supported by a Research Councils UK fellowship in Exabyte Informatics.

References

1. D. Alrajeh, O. Ray, A. Russo, and S. Uchitel. Extracting Requirements from Scenarios with ILP. In *Proc. 16th Int. Conf. on ILP*, volume 4455 of *LNCS*, pages 63–77. Springer, 2007.
2. M. Bain and S. Muggleton. Non-monotonic learning. In *Mach. Intel. 12*, pages 105–119. OUP, 1991.
3. K. L. Clark. Negation as failure rule. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.
4. Y. Dimopoulos and A. Kakas. Learning non-monotonic logic programs: Learning exceptions. In *Proc. 8th Europ. Conf. on Mach. Learn.*, volume 912 of *LNAI*, pages 122–138. Springer, 1995.
5. K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In *Proc. 6th Int. Conf. on Logic Programming*, pages 234–254. MIT Press, 1989.
6. F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy Theory Revision: Induction and Abduction in INTHELEX. *Mach. Learn.*, 38(1/2):133–156, 2000.
7. M. Nicosia G. Ruffo F. Bergadano, D. Gunetti. Learning logic programs with negation as failure. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 107–123. IOS Press, 1996.
8. L. Fogel and G. Zaverucha. Normal programs and multiple predicate learning. In *Proc. 8th Int. Workshop on ILP*, pages 175–184. Springer, 1998.
9. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Proc. 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.
10. K. Inoue and Y. Kudoh. Learning extended logic programs. In *Proc. 15th Int. Joint Conf. on AI*, volume I, pages 176–181. Morgan Kaufmann, 1997.
11. F. Jacob and J. Monod. Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3:318–356, 1961.
12. A. Kakas, R. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
13. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

14. E. Lamma, F. Riguzzi, and L. Pereira. Strategies in combined learning via logic programs. *Mach. Learn.*, 38(1-2):63–87, 2000.
15. J. Lloyd. *Foundations of Logic Programming*. Springer, 1987.
16. L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 219–235. IOS Press, 1996.
17. S. Moyle. Using theory completion to learn a robot navigation control program. In *Proc. 12th Int. Workshop on ILP*, volume 2583 of *LNAI*, pages 182–197. Springer, 2002.
18. S. Moyle. *An investigation into theory completion techniques in inductive logic programming*. PhD thesis, University of Oxford, UK, 2003.
19. S. Moyle and S. Muggleton. Learning programs in the event calculus. In *Proc. 7th Int. Workshop on ILP*, volume 1297 of *LNAI*, pages 205–212. Springer, 1997.
20. S. Muggleton. Inverse Entailment and Progol. *New Generation Computing*, 13(3-4):245–286, 1995.
21. S. Muggleton and C. Bryant. Theory Completion Using Inverse Entailment. In *Proc. 10th Int. Conf. on ILP*, volume 1866 of *LNCS*, pages 130–146. Springer, 2000.
22. S. Muggleton and L. De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
23. R. Otero. Induction of Stable Models. In *Proc. 11th Int. Conf. on ILP*, volume 2157 of *LNAI*, pages 193–205. Springer, 2001.
24. R. Otero. Embracing Causality in Inducing the Effects of Actions. In *Proc. 10th Conf. of the Spanish Association of AI*, volume 3040 of *LNAI*, pages 291–301. Springer, 2004.
25. O. Ray. Using abduction for induction of normal logic programs. In *Proc. ECAI'06 Workshop on Abduction and Induction in AI and Scientific Modelling*, pages 28–31, 2006.
26. O. Ray, K. Broda, and A. Russo. Hybrid Abductive Inductive Learning: a Generalisation of Progol. In *Proc. 13th Int. Conf. on ILP*, volume 2835 of *LNAI*, pages 311–328. Springer, 2003.
27. O. Ray, K. Broda, and A. Russo. A hybrid abductive inductive proof procedure. *Logic Journal of the IGPL*, 12(5):371–397, 2004.
28. O. Ray and A. Kakas. ProLogICA: a practical system for Abductive Logic Programming. In *Proc. 11th Int. Workshop on Non-monotonic Reasoning*, pages 304–312, 2006.
29. C. Sakama. Induction from answer sets in nonmonotonic logic programs. *ACM Transactions on Computational Logic*, 6(2):203–231, 2005.
30. J. Seitzer. Stable ILP: exploring the added expressivity of negation in the background knowledge. In *Proc. IJCAI'95 Workshop on Frontiers of ILP*, 1997.
31. M. Shanahan. *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press, 1997.
32. A. Srinivasan. *The Aleph Manual (version 4)*, 2003. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/index.html.

Equivalence Issues in Abduction and Induction

Chiaki Sakama[1] and Katsumi Inoue[2]

[1] Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp
[2] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
ki@nii.ac.jp

Abstract. This paper discusses several equivalence issues in abduction and induction. Three different problems: equivalence of background theories, equivalence of explanations, and equivalence of observations are considered in the context of first-order logic and nonmonotonic logic programming. Necessary and sufficient conditions for those problems, and computational complexity results are provided.

1 Introduction

Suppose a multiagent society where individual agents have their own knowledge bases. To solve problems cooperatively, agents must share their information in the society. It is likely to happen, however, that the same information is represented in different ways by each agent. To evaluate information contents and to identify different information sources, the notion of equivalence relation between theories/programs is utilized. The equivalence relation between theories/programs is also important in program development. Given a specification of a problem, a programmer transforms it into an executable program which would be further optimized to increase efficiency. In every step, a program is requested to be semantically equivalent to the original specification.

There is a number of ways for identifying different logical theories. In classical logic, two first-order theories are equivalent if they have the same logical consequences. In logic programming, two logic programs are equivalent if they have the same semantics [9]; and a stronger notion of equivalence is used under the name of *strong equivalence* [8]. These equivalence relations compare capabilities of deductive reasoning between theories/programs. On the other hand, considering intelligent agents that can perform commonsense reasoning, it is necessary to have a framework of comparing capabilities of *non-deductive* reasoning like abduction and induction. This motivates the studies by Inoue and Sakama [6, 7, 13] which introduce several criteria for identifying abductive/inductive theories/programs.

Abduction and induction have analogous inference mechanisms: they both produce hypotheses to explain observations using background theories [2]. Thus, there are at least three parameters in this task: background theories, explanations, and observations. Several equivalence issues are then considered:

Equivalence of background theories : Two background theories are considered equivalent if they produce the same explanations for any observation. This equivalence measure is useful for comparing "information contents" of different background theories.

Equivalence of explanations : Two explanations are considered equivalent if they account for the same observation under a given background theory. This equivalence measure is useful for comparing "explanation power" of different explanations.

Equivalence of observations : Two observations are considered equivalent if they produce the same explanations under a given background theory. This equivalence measure is useful for comparing "evidential power" of different observations.

The conditions for those equivalence issues generally depend on a logic on which abduction/induction is based. Moreover, those conditions differ among individual abduction/induction algorithms. Inoue and Sakama [6, 7] study the problem of equivalence of background theories in abduction under first-order logic and abductive logic programming. Sakama and Inoue [13] study the corresponding problem in induction and compare conditions for different algorithms in *inductive logic programming* (ILP). On the other hand, equivalence issues with respect to explanations/observations have not been studied so far.

The purpose of this paper is to discuss several equivalence issues for abduction and induction. We first review the results of [6, 7, 13] on the equivalence of background theories in Section 2. We then investigate the remaining two problems; equivalence of explanations in Section 3 and equivalence of observations in Section 4. We provide results under two logics, first-order logic and (nonmonotonic) logic programming, which are two most popular logics used in the literature of abduction/induction. Section 5 analyzes the results of this paper, Section 6 summarizes the paper.

2 Equivalence of Background Theories

2.1 Abductive Equivalence in First-Order Logic

In this section, we first consider the case that the underlying logic is first-order logic. As stated in Section 1 we capture both abduction and induction as a process of hypothesis generation given a background theory and observations. To understand two inference mechanisms in a unified framework, we define abduction in a general setting.

Definition 2.1. (**abductive theory**) An *abductive (first-order) theory* is defined as a pair (B, H) where B and H are sets of first-order formulas, respectively representing a *background theory* and a *candidate hypothesis*. An abductive theory is called *propositional* if both B and H are finite propositional theories.

Let O be any formula representing an *observation*. Then, a set $E \subseteq H$ is an *explanation* of O if

- $B \cup E \models O$, and
- $B \cup E$ is consistent.

An explanation is called *ground* if it contains no variable.

Note that the above definition also characterizes *induction*. A typical induction problem is: given a finite set G of observations and a background knowledge B, find a hypothesis E such that $B \cup E \models G$ where $B \cup E$ is consistent.[3] A finite set G of observation is represented as a single formula $O = \bigwedge_{g \in G} g$. When candidate hypothesis H is not specified in advance, we can put $H = \mathcal{F}$ with the set $\mathcal{F}$ of all first-order formulas in the language. So in this paper we discuss equivalence issues in abduction hereafter, but the same results hold for induction as well.

Remark: In induction problems, *negative observations* are often considered as well as positive ones. For any negative observation G, the condition $B \cup E \not\models G$ is requested for any explanation E. To handle negative observations, the notion of *anti-explanations* in the context of *extended abduction* is usable [4]. Equivalence problems in this paper are extended to handle negative observations as well. For simplicity reasons, we handle positive observations only in this paper.[4]

To argue equivalence issues in abductive logic, Inoue and Sakama [6] introduce two different criteria of abductive equivalence.

Definition 2.2. (**explainable equivalence**) Two abductive theories (B_1, H_1) and (B_2, H_2) are *explainably equivalent* if, for any observation O, there is an explanation of O in (B_1, H_1) iff there is an explanation of O in (B_2, H_2).

Definition 2.3. (**explanatory equivalence**) Two abductive theories (B_1, H_1) and (B_2, H_2) are *explanatorily equivalent* if, for any observation O, there is an explanation E_1 of O in (B_1, H_1) iff there is an explanation E_2 of O in (B_2, H_2) such that $E_1 \equiv E_2$.

Explainable equivalence requires that two abductive theories have the same *explainability* for any observation. By contrast, explanatory equivalence assures that two abductive theories have the same *explanation contents* for any observation. Explanatory equivalence is stronger than explainable equivalence and the former implies the latter.

Example 2.1. Consider two abductive theories (B_1, H_1) and (B_2, H_2) such that

$$
\begin{aligned}
B_1 :\ & grass\_is\_wet \supset shoes\_are\_wet, \\
& rained\_last\_night \supset grass\_is\_wet, \\
& sprinkler\_was\_on \supset grass\_is\_wet. \\
H_1 :\ & rained\_last\_night,\ \ sprinkler\_was\_on\,.
\end{aligned}
$$

$$
\begin{aligned}
B_2 :\ & grass\_is\_wet \supset shoes\_are\_wet, \\
& rained\_last\_night \supset grass\_is\_wet. \\
H_2 :\ & rained\_last\_night,\ \ sprinkler\_was\_on.
\end{aligned}
$$

Then, (B_1, H_1) and (B_2, H_2) are explainably equivalent, but not explanatory equivalent. That is, every observation explainable in (B_1, H_1) is also explainable in (B_2, H_2),

[3] This type of induction is called *explanatory induction* [2].

[4] Equivalence of background theories in extended abduction is discussed in [7].

and vice versa. On the other hand, the observation $O = shoes\_are\_wet$ has the explanation $E = \{ sprinkler\_was\_on \}$ in (B_1, H_1), but E does not explain O in (B_2, H_2).

In the above example, (B_1, H_1) has the cause-effect knowledge $sprinkler\_was\_on \supset grass\_is\_wet$, but (B_2, H_2) does not. This means that if it is later known that it was not rained last night ($\neg\, rained\_last\_night$), (B_2, H_2) cannot explain the observation $O = shoes\_are\_wet$ anymore. As a result, $(B_1 \cup \{ \neg\, rained\_last\_night \}, H_1)$ and $(B_2 \cup \{\neg\, rained\_last\_night\}, H_2)$ are not explainably equivalent.

Thus, two equivalence relations compare explanation power of abductive theories in different ways. Inoue and Sakama [6] provide necessary and sufficient conditions for each equivalence relation. In the following, $Th(\Sigma)$ denotes the set of logical consequences of a set Σ of first-order formulas.

Definition 2.4. **(extension)** Let (B, H) be an abductive theory. An *extension* of (B, H) is defined as $Th(B \cup S)$ where S is a maximal subset of H such that $B \cup S$ is consistent. The set of all extensions of (B, H) is denoted by $Ext(B, H)$.

Theorem 2.1. *[6] Let (B_1, H_1) and (B_2, H_2) be two abductive theories. Then,*

1. *(B_1, H_1) and (B_2, H_2) are explainably equivalent iff $Ext(B_1, H_1) = Ext(B_2, H_2)$.*
2. *(B_1, H_1) and (B_2, H_2) are explanatorily equivalent iff $B_1 \equiv B_2$ and $H_1' = H_2'$ where $H_i' = \{\, h \in H_i \mid B_i \cup \{h\}$ is consistent $\}$ for $i = 1, 2$.*

Since any element in $H_i \setminus H_i'$ is of no use for explaining observations, two abductive theories are assumed to have a common hypothesis set H for explanatory equivalence.

The next theorem states the computational complexity of each equivalence problem.[5]

Theorem 2.2. *[6]*

1. *Deciding explainable equivalence of two (propositional) abductive theories is Π_2^P-complete.*
2. *Deciding explanatory equivalence of two (propositional) abductive theories is coNP-complete.*

2.2 Abductive Logic Programming

Next, we consider the case that the underlying logic is *abductive logic programming* (ALP) [1]. In contrast to first-order logic, in ALP a background theory is given as a *nonmonotonic logic program* in general.

A logic program considered in this paper is the class of *general extended disjunctive program* (GEDP) [5], which is a set of rules of the form:

$$L_1 ; \cdots ; L_k ; not\, L_{k+1} ; \cdots ; not\, L_l \leftarrow L_{l+1}, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$$

[5] Throughout the paper, complexity results are stated in terms of the size of background theories and candidate hypotheses, unless stated otherwise.

$(n \geq m \geq l \geq k \geq 0)$, where each L_i is a positive/negative literal, i.e., A or $\neg A$ for an atom A, and not is *negation as failure*. $not\ L$ is called an *NAF-literal*. The symbol ; represents disjunction. The left-hand side of the rule is the *head*, and the right-hand side is the *body*. For each rule r of the above form, $head^+(r)$, $head^-(r)$, $body^+(r)$ and $body^-(r)$ denote the sets of literals $\{L_1, \ldots, L_k\}$, $\{L_{k+1}, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, and $\{L_{m+1}, \ldots, L_n\}$, respectively. Also, $not\_head^-(r)$ and $not\_body^-(r)$ denote the sets of NAF-literals $\{not\ L_{k+1}, \ldots, not\ L_l\}$ and $\{not\ L_{m+1}, \ldots, not\ L_n\}$, respectively. A disjunction/conjunction of (NAF-)literals in a rule is identified with its corresponding sets of (NAF-)literals. A rule r is often written as

$$head^+(r)\,;\, not\_head^-(r) \leftarrow body^+(r),\, not\_body^-(r)$$

or $head(r) \leftarrow body(r)$ where $head(r) = head^+(r) \cup not\_head^-(r)$ and $body(r) = body^+(r) \cup not\_body^-(r)$. A rule $L \leftarrow$ is identified with a literal L. A program P is *basic* if $head^-(r) = body^-(r) = \emptyset$ for every rule r in P. A program P is an *extended disjunctive program* (EDP) if $head^-(r) = \emptyset$ for every rule r in P. A program, rule, or literal is *ground* if it contains no variable. A program P with variables is a shorthand of its *ground instantiation* $Ground(P)$, the set of ground rules obtained from P by substituting variables in P by elements of its Herbrand universe in every possible way.

A semantics of a GEDP is given by the *answer set semantics* [3, 5]. Let Lit be the set of all ground literals in the language of a program. Suppose a program P and a set of literals $S(\subseteq Lit)$. Then, the *reduct* P^S is the program which contains the ground rule $head^+(r) \leftarrow body^+(r)$ iff there is a rule r in $Ground(P)$ such that $head^-(r) \setminus S = \emptyset$ and $body^-(r) \cap S = \emptyset$. Given a basic program P, $Cn(P)$ denotes the smallest set of ground literals which is (i) *closed* under P, i.e., for every ground rule $head^+(r) \leftarrow body^+(r)$ in $Ground(P)$, $body^+(r) \subseteq Cn(P)$ implies $head^+(r) \cap Cn(P) \neq \emptyset$; and (ii) *logically closed*, i.e., it is either consistent or equal to Lit. Given a GEDP P and a set S of literals, S is an *answer set* of P if $S = Cn(P^S)$. A program has none, one, or multiple answer sets in general. An answer set is *consistent* if it is not Lit. A program is *consistent* if it has a consistent answer set; otherwise it is inconsistent. Throughout this paper, a program is assumed to be consistent unless stated otherwise. The set of all answer sets of a program P is denoted by $AS(P)$. When P is an EDP, $AS(P)$ becomes an *anti-chain* set, i.e., no element $S \in AS(P)$ is a proper subset of another element $T \in AS(P)$. This is not the case for GEDPs in general (see Example 2.2). A conjunction $L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$ of ground (NAF-)literals is satisfied in an answer set S if $\{ L_1, \ldots, L_m \} \subseteq S$ and $\{ L_{m+1}, \ldots, L_n \} \cap S = \emptyset$.

A literal L is a consequence of *skeptical reasoning* (resp. *credulous reasoning*) in a program P if L is included in every (resp. some) answer set of P. For a consistent program P, define

$$skp(P) = \bigcap_{S \in AS(P)} S \quad \text{and} \quad crd(P) = \bigcup_{S \in AS(P)} S.$$

Clearly, $skp(P) \subseteq crd(P)$ holds for any consistent program P.

Two programs P_1 and P_2 are (*weakly*) *equivalent* if $AS(P_1) = AS(P_2)$. P_1 and P_2 are *strongly equivalent* if $AS(P_1 \cup Q) = AS(P_2 \cup Q)$ for any program Q [8]. By the definition, strong equivalence implies weak equivalence, but not vice versa. Given

a set R of rules, P_1 and P_2 are strongly equivalent *with respect to* R if $AS(P_1 \cup Q) = AS(P_2 \cup Q)$ for any $Q \subseteq R$ (this equivalence is called *relative equivalence*). Strong equivalence is a special case of relative equivalence where R is the set of all rules in the language.

Proposition 2.3 *Let P_1 and P_2 be two programs. If P_1 and P_2 are (weakly) equivalent, $skp(P_1) = skp(P_2)$ and $crd(P_1) = crd(P_2)$.*

The converse of the above proposition does not hold in general.

Example 2.2. Consider two programs:

$$\begin{aligned} P_1 :\ & p \leftarrow not\, q, \\ & q \leftarrow not\, p. \\ P_2 :\ & p \leftarrow q, \\ & q\,;\, not\, q \leftarrow \end{aligned}$$

where $AS(P_1) = \{\{p\}, \{q\}\}$ and $AS(P_2) = \{\emptyset, \{p, q\}\}$. Then, $skp(P_1) = skp(P_2) = \emptyset$ and $crd(P_1) = crd(P_2) = \{p, q\}$.

An *abductive program* is defined as a pair $\langle P, \mathcal{A} \rangle$ where P and $\mathcal{A}$ are GEDPs respectively representing a background theory and a candidate hypothesis. In particular, an abductive program $\langle P, \mathcal{A} \rangle$ is called an *abductive EDP* if both P and $\mathcal{A}$ are EDPs. An abductive program $\langle P, \mathcal{A} \rangle$ is called an *abductive definite program* if both P and $\mathcal{A}$ are definite logic programs, i.e., sets of definite Horn clauses. Any element in $\mathcal{A}$ is called an *abducible*. An abductive program $\langle P, \mathcal{A} \rangle$ is called *propositional* if both P and $\mathcal{A}$ are finite ground programs.

Definition 2.5. (**belief sets**) Let $\langle P, \mathcal{A} \rangle$ be an abductive program. For any $E \subseteq \mathcal{A}$, a consistent answer set S of $P \cup E$ is called a *belief set* of $\langle P, \mathcal{A} \rangle$ (with respect to E).

In abductive logic programming, we define an *observation* as a ground literal.

Definition 2.6. (**credulous/skeptical explanation**) Let $\langle P, \mathcal{A} \rangle$ be an abductive program, and O an observation. A set $E \subseteq \mathcal{A}$ is a *credulous explanation* of O in $\langle P, \mathcal{A} \rangle$ if O is included in some belief set of $\langle P, \mathcal{A} \rangle$ (with respect to E). A set $E \subseteq \mathcal{A}$ is a *skeptical explanation* of O in $\langle P, \mathcal{A} \rangle$ if O is included in every belief set of $\langle P, \mathcal{A} \rangle$ (with respect to E).

A credulous/skeptical explanation is called *ground* if it contains no variable. Abduction for credulous/skeptical explanations is also called *credulous/skeptical abduction*.

Remark: In the literature of abductive logic programming, abducibles are usually restricted to (ground) literals. Any abductive program $\langle P, \mathcal{A} \rangle$ which contains rules in $\mathcal{A}$ is transformed to a semantically equivalent abductive program in which abducibles contain only literals. Given an abductive program $\langle P, \mathcal{A} \rangle$, let

$$\begin{aligned} P' &= (P \setminus \mathcal{A}) \ \cup\ \{\, head(r) \leftarrow body(r),\, N_r \mid r \in \mathcal{A} \,\} \ \cup\ \{\, N_r \leftarrow \mid r \in \mathcal{A} \cap P \,\}, \\ \mathcal{A}' &= \{\, N_r \mid r \in \mathcal{A} \,\}, \end{aligned}$$

where N_r is a newly introduced atom (called the *name* of r) uniquely associated with each rule r in $\mathcal{A}$. With this setting, for any observation there is a 1-1 correspondence between explanations in $\langle P, \mathcal{A} \rangle$ and those in $\langle P', \mathcal{A}' \rangle$ [12].

Without loss of generality, we assume an observation O as a ground literal. If an observation O is given as a conjunction of ground (NAF-)literals to an abductive program $\langle P, \mathcal{A} \rangle$, O is transformed to the rule $G \leftarrow O$ where G is a new ground atom appearing nowhere in $P \cup \mathcal{A}$. With this setting, O is satisfied by an answer set of $P \cup E$ for $E \subseteq \mathcal{A}$ iff G is included in an answer set of $P \cup \{ G \leftarrow O \} \cup E$. Then G has an explanation in an abductive program $\langle P \cup \{ G \leftarrow O \}, \mathcal{A} \rangle$.

Example 2.3. Let $\langle P, \mathcal{A} \rangle$ be an abductive program such that

$$\begin{aligned} P:\ & watch\text{-}TV\,;\, sleeping \leftarrow holiday,\ not\ busy\,, \\ & working \leftarrow holiday,\ busy\,, \\ & holiday \leftarrow\ . \\ \mathcal{A}:\ & busy\,. \end{aligned}$$

The observation $O_1 = watch\text{-}TV$ has the empty set $E_1 = \emptyset$ as the credulous explanation. The observation $O_2 = working$ has the skeptical explanation $E_2 = \{ busy \}$.

Definition 2.7. (**explainable/explanatory equivalence**) Two abductive logic programs $\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ are *explainably equivalent* if, for any observation O, there is a credulous/skeptical explanation of O in $\langle P_1, \mathcal{A}_1 \rangle$ iff there is a credulous/skeptical explanation of O in $\langle P_2, \mathcal{A}_2 \rangle$. On the other hand, $\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ are *explanatorily equivalent* if, for any observation O, there is a credulous/skeptical explanation E of O in $\langle P_1, \mathcal{A}_1 \rangle$ iff there is a credulous/skeptical explanation E of O in $\langle P_2, \mathcal{A}_2 \rangle$.

Note that in logic programming the meaning of rules depends on their syntax in general. So two explanations $E_1 = \{ p \leftarrow \neg q \}$ and $E_2 = \{ q \leftarrow \neg p \}$ have different meanings. This is in contrast to the case of first-order abduction.

In [6] necessary and sufficient conditions for explainable/explanatory equivalence in credulous abduction are given. In [13] a necessary and sufficient condition for explanatory equivalence in skeptical abduction is given in the context of *inductive logic programming* (ILP). We summarize those results below, together with new results.

Theorem 2.4. *Let $\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ be two abductive programs. Then,*

1. *$\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ are explainably equivalent in credulous abduction iff*

$$\bigcup_{E \in \mathcal{A}_1} crd(P_1 \cup E) \ = \bigcup_{F \in \mathcal{A}_2} crd(P_2 \cup F)$$

   where $P_1 \cup E$ and $P_2 \cup F$ are consistent.

2. *$\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ are explainably equivalent in skeptical abduction iff there is a set $E \subseteq \mathcal{A}_1$ and $F \subseteq \mathcal{A}_2$ satisfying*

$$skp(P_1 \cup E) = skp(P_2 \cup F)$$

   where $P_1 \cup E$ and $P_2 \cup F$ are consistent.

3. *$\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ with $\mathcal{A}_1 = \mathcal{A}_2 = \mathcal{A}$ are explanatorily equivalent in credulous/skeptical abduction iff P_1 and P_2 are strongly equivalent with respect to $\mathcal{A}$.*

Proof. The results of 1 and 3 are due to [6, 13]. Here we show 2. By the definition, $\langle P_1, \mathcal{A}_1 \rangle$ and $\langle P_2, \mathcal{A}_2 \rangle$ are explainably equivalent in skeptical abduction iff for any observation O, $O \in skp(P_1 \cup E)$ with some $E \subseteq \mathcal{A}_1$ such that $P_1 \cup E$ is consistent and $O \in skp(P_2 \cup F)$ with some $F \subseteq \mathcal{A}_2$ such that $P_2 \cup F$ is consistent. Hence, the result holds. □

Theorem 2.5. *1. Deciding explainable equivalence of two (propositional) abductive programs is Π_2^P-hard in both credulous and skeptical abduction.*

2. Deciding explanatory equivalence of two (propositional) abductive programs is Π_2^P-complete in both credulous and skeptical abduction.

Proof. The result of (1) in credulous abduction is due to [6]. To see the result in skeptical abduction, the problem contains the case that the abducibles are empty. In this case, the problem reduces to deciding equivalence of skeptical consequences between two background programs. This is done by checking whether any literal is included in every answer set of two programs. This is known as a Π_2^P-complete task [5]. Hence, the result holds. The result of (2) in credulous abduction is due to [6]. Since the necessary and sufficient condition of explanatory equivalence in skeptical abduction is the same as that of credulous abduction by Theorem 2.4(3), the result holds. □

3 Equivalence of Explanations

Next we turn our attention to the problem of equivalence of explanations.

Definition 3.1. (**equivalent explanation**) Given an abductive theory (B, H), two explanations E_1 and E_2 are *equivalent* if, for any observation O, E_1 is an explanation of O in (B, H) iff E_2 is an explanation of O in (B, H).

The notion of equivalent explanations provides a method for identifying different explanations which are abduced for an arbitrary observation in a background knowledge. The next result holds for first-order abduction.

Theorem 3.1. *Let (B, H) be an abductive theory. Then, two explanations E_1 and E_2 are equivalent iff $B \cup E_1 \equiv B \cup E_2$.*

Proof. E_1 and E_2 are equivalent
iff $B \cup E_1 \models O \Leftrightarrow B \cup E_2 \models O$ for any formula O
iff $B \cup E_1 \equiv B \cup E_2$. □

The above theorem states that in first-order abduction explanations are equivalent if and only if $B \cup E_1$ and $B \cup E_2$ are logically equivalent. In other words, logically different formulas can become an equivalent explanation depending on the context of B.

Example 3.1. Given $B = \{\, p \supset q,\ \ q \supset p,\ \ p \wedge q \supset r \,\}$ and $H = \{\, p,\, q \,\}$, $E_1 = \{\, p \,\}$, $E_2 = \{\, q \,\}$, and $E_3 = \{\, p,\, q \,\}$ are all equivalent explanations.

In the context of abductive logic programming, the notion of equivalent explanation is defined in the same manner by replacing an abductive theory (B, H) with an abductive program $\langle\, P, \mathcal{A} \,\rangle$ in Definition 3.1. Then, the following result holds.

Theorem 3.2. *Let $\langle\, P, \mathcal{A} \,\rangle$ be an abductive program. Then, two explanations E_1 and E_2 are equivalent if $AS(P \cup E_1) = AS(P \cup E_2)$ where $P \cup E_1$ and $P \cup E_2$ are consistent. The only-if part also holds for credulous abduction if $\langle\, P, \mathcal{A} \,\rangle$ is an abductive EDP.*[6]

Proof. The if part is obvious for both skeptical and credulous abduction. We show the only-if part for credulous abduction. Suppose that $AS(P \cup E_1) \neq AS(P \cup E_2)$ and there is a consistent answer set S such that $S \in AS(P \cup E_1) \setminus AS(P \cup E_2)$. (a) If there is a finite subset $G \subseteq S$ such that $G \not\subseteq T$ for any $T \in AS(P \cup E_2)$, E_1 explains G but E_2 does not. This contradicts the assumption that E_1 and E_2 are equivalent. (b) Otherwise, for every finite subset $G \subseteq S$, $G \subseteq T$ for some $T \in AS(P \cup E_2)$. Then, $S \subseteq T$ holds. (Otherwise, there is a literal $L \in S \setminus T$. Then, for a finite subset $G' \subseteq S$ such that $L \in G'$, $G' \not\subseteq T$. Contradiction.) As $S \notin AS(P \cup E_2)$, $S \subset T$ holds (†). Since $AS(P \cup E_1)$ is an anti-chain set, $T \notin AS(P \cup E_1)$. Then, $T \in AS(P \cup E_2) \setminus AS(P \cup E_1)$. (c) If there is a finite subset $G' \subseteq T$ such that $G' \not\subseteq S'$ for any $S' \in AS(P \cup E_1)$, E_2 explains G' but E_1 does not. This contradicts the equivalence assumption of E_1 and E_2. (d) Otherwise, for every finite subset $G' \subseteq T$, $G' \subseteq S'$ for some $S' \in AS(P \cup E_1)$. Then, $T \subseteq S'$ holds as above. Since $T \notin AS(P \cup E_1)$, $T \subset S'$ holds. By (†), $S \subset S'$ holds for two answer sets S and S' of $P \cup E_1$. But this is impossible, since $AS(P \cup E_1)$ is an anti-chain set. □

Example 3.2. Let $\langle\, P, \mathcal{A} \,\rangle$ be an abductive program such that

$$
\begin{aligned}
P:\ & p \leftarrow not\, q, \\
& q \leftarrow not\, p, \\
& r \leftarrow not\, r. \\
\mathcal{A}:\ & r \leftarrow p, \\
& r \leftarrow not\, q.
\end{aligned}
$$

Then, $E_1 = \{\, r \leftarrow p \,\}$, $E_2 = \{\, r \leftarrow not\, q \,\}$ and $E_3 = \{\, r \leftarrow p,\ \ r \leftarrow not\, q \,\}$ are equivalent in both skeptical/credulous abduction.

In skeptical abduction, the condition $AS(P \cup E_1) = AS(P \cup E_2)$ is not necessary for the equivalence of explanations. For instance, in the abductive program $\langle\, P, \mathcal{A} \,\rangle$ where $P = \emptyset$ and $\mathcal{A} = \{\, p\,;\, q \leftarrow,\ \ \ r\,;\, s \leftarrow \,\}$, two skeptical explanations $E_1 = \{\, p\,;\, q \leftarrow \,\}$ and $E_2 = \{\, r\,;\, s \leftarrow \,\}$ are equivalent but $AS(P \cup E_1) \neq AS(P \cup E_2)$. A necessary condition for the equivalence of skeptical explanations is $skp(P \cup E_1) = skp(P \cup E_2)$, which directly follows by the definition.

[6] The only-if part holds for GEDPs when a program has a finite number of answer sets. At the moment, however, we do not complete the proof for GEDPs having infinite answer sets in general.

Deciding logical equivalence of two propositional theories is a coNP-complete task. The equivalence $AS(P \cup E_1) = AS(P \cup E_2)$ is known by testing the (weak) equivalence of two (propositional) programs $P \cup E_1$ and $P \cup E_2$, which is Π_2^P-complete [10]. The task of deciding whether a literal is a skeptical consequence of a propositional GEDP is also Π_2^P-complete [5]. These facts imply the following results.

Theorem 3.3. *The following complexity results holds with respect to the size of background theories and explanations.*

1. *Deciding equivalence of two ground explanations in a (propositional) abductive theory is coNP-complete.*
2. *Deciding equivalence of two ground explanations in a (propositional) abductive program is Π_2^P-complete in both credulous and skeptical abduction.*
3. *Deciding equivalence of two ground explanations in a (propositional) abductive definite program is done in polynomial time.*

4 Equivalence of Observations

We finally consider the problem of equivalence of observations.

Definition 4.1. (**equivalent observation**) Given an abductive theory (B, H), two observations O_1 and O_2 are *equivalent* if, for any $E \subseteq H$, E is an explanation of O_1 in (B, H) iff E is an explanation of O_2 in (B, H).

The notion of equivalent observations provides a method for identifying different evidences which are used for abduction.

Theorem 4.1. *Let (B, H) be an abductive theory. Then, two observations O_1 and O_2 are equivalent iff $B \cup E \models O_1 \equiv O_2$ for any $E \subseteq H$ such that $B \cup E$ is consistent.*

Proof. O_1 and O_2 are equivalent
iff $B \cup E \models O_1 \Leftrightarrow B \cup E \models O_2$ for any $E \subseteq H$ such that $B \cup E$ is consistent
iff $B \cup E \models O_1 \equiv O_2$ for any $E \subseteq H$ such that $B \cup E$ is consistent. □

The result shows that equivalence of observations depends on an abductive theory.

Example 4.1. Given $(B_1, H_1) = (\{\, p \supset q \,\}, \{\, p, q \,\})$, $O_1 = p$ and $O_2 = p \wedge q$ are equivalent. On the other hand, O_1 and O_2 are not equivalent in $(B_2, H_2) = (\{\, p \vee q \,\}, \{\, p, q \,\})$.

In the above example, the equivalence of O_1 and O_2 implies that the additional evidence q in O_2 has no effect in constructing explanations in (B_1, H_1).

The notion of equivalence of observations is defined for abductive logic programming by replacing an abductive theory (B, H) with an abductive program $\langle P, \mathcal{A} \rangle$ in Definition 4.1.

Theorem 4.2. *Let $\langle P, \mathcal{A} \rangle$ be an abductive program.*

1. *Two observations O_1 and O_2 are equivalent in credulous abduction iff*

$$O_1 \in crd(P \cup E) \ \textit{ and } \ O_2 \in crd(P \cup E)$$

   for any $E \subseteq \mathcal{A}$ such that $P \cup E$ is consistent.

2. *Two observations O_1 and O_2 are equivalent in skeptical abduction iff*

$$O_1 \in skp(P \cup E) \ \textit{ and } \ O_2 \in skp(P \cup E)$$

   for any $E \subseteq \mathcal{A}$ such that $P \cup E$ is consistent.

Proof. In credulous abduction, O_1 and O_2 are equivalent if O_1 (resp. O_2) is included in some consistent answer set S (resp. T) of $P \cup E$ for any $E \subseteq \mathcal{A}$. Then, O_1 and O_2 are included in the set of credulous consequences of $P \cup E$. In skeptical abduction, O_1 and O_2 are equivalent if both O_1 and O_2 are included in every consistent answer set of $P \cup E$ for any $E \subseteq \mathcal{A}$. Then, O_1 and O_2 are included in the set of skeptical consequences of $P \cup E$. □

Example 4.2. Let $\langle P, \mathcal{A} \rangle$ be an abductive program such that

$$\begin{aligned} P: \ & wet \leftarrow rain, \ not \ \neg wet, \\ & \neg wet \leftarrow rain, \ not \ wet. \\ \mathcal{A}: \ & rain. \end{aligned}$$

Then, $O_1 = wet$ and $O_2 = \neg wet$ are equivalent in credulous abduction, but not equivalent in skeptical abduction.

In the above example, the equivalence of O_1 and O_2 presents a situation that wet or $\neg wet$ could equally happen on the same ground $rain$.

An abductive program is transformed to a semantically equivalent GEDP [5]. The task of deciding whether a literal is a credulous (resp. skeptical) consequence of a propositional GEDP is Σ_2^P-complete (resp. Π_2^P-complete) [5].

Theorem 4.3. *The following complexity results hold with respect to the size of background theories and candidate hypotheses.*

1. *Deciding equivalence of two observations in a (propositional) abductive theory is coNP-complete.*
2. *Deciding equivalence of two observations in a (propositional) abductive program is Σ_2^P-complete in credulous abduction and Π_2^P-complete in skeptical abduction.*

5 Discussion

In this section, we compare conditions in different equivalence issues. First, recall the problem of equivalence of background theories. In first-order abduction, explanatory

Table 1. Computational Complexity

Logic	Background Theories (explainable / explanatory)	Explanations	Observations
FOL (propositional)	Π_2^P-complete / coNP-complete	coNP-complete	
ALP (credulous)	Π_2^P-hard / Π_2^P-complete	Π_2^P-complete	Σ_2^P-complete
(skeptical)	Π_2^P-hard / Π_2^P-complete	Π_2^P-complete	

equivalence is stronger than explainable equivalence, so that the condition of explanatory equivalence is much restrictive; it requires the logical equivalence of two background theories (Theorem 2.1). By contrast, the task of deciding explanatory equivalence of two abductive theories is easier than the task of deciding explainable equivalence (Theorem 2.2). In abductive logic programming, explainable equivalence is checked by comparing credulous/skeptical consequences that are computed by belief sets of each abductive program (Theorem 2.4). Since belief sets are computable using proof procedures of answer set programming, checking explainable equivalence is done on existing answer set solvers. To guarantee explanatory equivalence, on the other hand, relative equivalence of two abductive programs is required. At the moment, no sophisticated procedure is known for testing relative equivalence. Computational cost of checking each equivalence is generally expensive (Theorem 2.5). As a special case, however, explainable equivalence of two *definite* abductive programs, which has a background theory as a definite logic program, can be decided in polynomial time [6]. This would be good news for existing ILP systems in which background theories are usually given as definite logic programs.

Second, consider the problem of equivalence of explanations. In first-order abduction, the problem is identical to judging logical equivalence of two theories (Theorem 3.1). In abductive logic programming, the problem is identical to the (weak) equivalence of two programs under some restriction on the syntax of a program (Theorem 3.2). Checking equivalence of explanations in abductive logic programming is generally harder than first-order abduction (Theorem 3.3). In the problem of equivalence of observations, it is identical to testing the logical equivalence of observations under an abductive theory in first-order abduction (Theorem 4.1). In abductive logic programming, comparison of skeptical/credulous consequences is requested (Theorem 4.2). It is observed that abductive logic programming is again harder than first-order abduction in general (Theorem 4.3). The complexity results are summarized in Table 1.

The equivalence problems considered in this paper also imply the *equivalence of predictions* which are deductive consequences of explanations in a background theory. Formally, given an abductive theory (B, H), if $B \cup E \models G$ for some $E \subseteq H$ such that $B \cup E$ is consistent, G is called a *prediction* of E in B. When two abductive theories (or abductive programs) are explainably equivalent, they have the same accountability for any prediction. By contrast, when two abductive theories are explanatory equivalent, they produce the same predictions by any explanation. On the other hand, given an abductive theory, if two explanations are equivalent, they produce the same predic-

tions in the background theory. Thus, different equivalence relations characterize the equivalence of predictions from different viewpoints.

Comparing first-order abduction and abductive logic programming, logical equivalence characterizes each problem in first-order abduction. In abductive logic programming, on the other hand, strong equivalence characterizes explanatory equivalence of background theories, (weak) equivalence characterizes equivalence of explanations, and equivalence of credulous/skeptical consequences characterizes explainable equivalence of background theories and equivalence of observations. Thus, different types of equivalence notions are used in different problems. What makes comparison of abductive programs more complicated is *nonmonotonicity* in abductive logic programming, which also makes computational task of equivalence testing harder than first-order abduction in general.

6 Conclusion

In this paper, we studied different types of equivalences in abduction and induction: explainable/explanatory equivalence of background theories, equivalence of explanations, and equivalence of observations. In each case, necessary and sufficient conditions for equivalence as well as computational complexity results are given, under both classical logic and nonmonotonic logic programming. These results shed light on the equivalence issue in non-deductive reasoning and apply to a general hypothetico-deductive framework common to both abduction and induction.

The results of this paper also have an important implication in program development in abductive/inductive logic programming. For instance, it is known that some basic transformations, such as *unfolding/folding*, do not preserve strong equivalence of logic programs [11]. This fact, together with the result of this paper, implies that such basic program transformations are *not* applicable to optimize background theories in ALP/ILP. If applied, explanations of abduction/induction may change in general. On the other hand, those transformations which preserve weak equivalence of logic programs are used for knowing explainability, since they preserve answer sets of a program and will not change credulous/skeptical consequences. Program development and optimization issues for abductive/inductive theories/programs are left for future research.

References

1. M. Denecker and A. Kakas. Abductive logic programming. In: A. C. Kakas and F. Sadri (eds.), *Computational Logic: Logic Programming and Beyond — Essays in Honour of Robert A. Kowalski, Part I, Lecture Notes in Artificial Intelligence*, vol. 2407, pp. 402–436, Springer-Verlag, 2002.
2. P. A. Flach and A. C. Kakas (eds.). *Abduction and Induction — Essays on their Relation and Integration*, Kluwer Academic, 2000.
3. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385, 1991.
4. K. Inoue and C. Sakama. Abductive framework for nonmonotonic theory change. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 204–210, Morgan Kaufmann, 1995.

5. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming* 35:39–78, 1998.
6. K. Inoue and C. Sakama. Equivalence in abductive logic. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pp. 472–477, 2005.
7. K. Inoue and C. Sakama. On abductive equivalence. In: Lorenzo Magnani (ed.), *Model-Based Reasoning in Science and Engineering: Cognitive Science, Epistemology, Logic*. *Studies in Logic*, pp. 333–352, College Publications, London, 2006.
8. V. Lifschitz, D. Pearce and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic* 2:526–541, 2001.
9. M. J. Maher. Equivalence of logic programs. In: J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 627–658, Morgan Kaufmann, 1988.
10. E. Oikarinen and T. Janhunen. Verifying the equivalence of logic programs in the disjunctive case. In: *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, *Lecture Notes in Artificial Intelligence*, vol. 2923, pp. 180–193, Springer-Verlag, 2004.
11. M. Osorio, J. A. Navarro, and J. Arrazola. Equivalence in answer set programming. In: *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation*, Lecture Notes in Computer Science, vol. 2372, pp. 57–75, Springer-Verlag, 2001.
12. C. Sakama and K. Inoue. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming* 3(6):671–715, 2003.
13. C. Sakama and K. Inoue. Inductive equivalence of logic programs. In: *Proceedings of the 15th International Conference on Inductive Logic Programming, Lecture Notes in Artificial Intelligence*, vol. 3625, pp. 312–329, Springer-Verlag, 2005.