Artificial Intelligence 259 (2018) 110–146

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The complexity and generality of learning answer set programs

Mark Law *, Alessandra Russo, Krysia Broda

Department of Computing, Imperial College London, SW7 2AZ, United Kingdom

ARTICLE INFO

Article history:
Received 2 September 2016
Received in revised form 20 February 2018
Accepted 15 March 2018
Available online 21 March 2018

Keywords:
Non-monotonic logic-based learning
Answer Set Programming
Complexity of non-monotonic learning

ABSTRACT

Traditionally most of the work in the field of Inductive Logic Programming (ILP) has addressed the problem of learning Prolog programs. On the other hand, Answer Set Programming is increasingly being used as a powerful language for knowledge representation and reasoning, and is also gaining increasing attention in industry. Consequently, the research activity in ILP has widened to the area of Answer Set Programming, witnessing the proposal of several new learning frameworks that have extended ILP to learning answer set programs. In this paper, we investigate the theoretical properties of these existing frameworks for learning programs under the answer set semantics. Specifically, we present a detailed analysis of the computational complexity of each of these frameworks with respect to the two decision problems of deciding whether a hypothesis is a solution of a learning task and deciding whether a learning task has any solutions. We introduce a new notion of *generality* of a learning framework, which enables us to define a framework to be more general than another in terms of being able to *distinguish* one ASP hypothesis solution from a set of incorrect ASP programs. Based on this notion, we formally prove a generality relation over the set of existing frameworks for learning programs under answer set semantics. In particular, we show that our recently proposed framework, *Context-dependent Learning from Ordered Answer Sets*, is more general than brave induction, induction of stable models, and cautious induction, and maintains the same complexity as cautious induction, which has the highest complexity of these frameworks.

1. Introduction

Over the last two decades there has been a growing interest in Inductive Logic Programming (ILP) [1], where the goal is to learn a logic program called a *hypothesis*, which together with a given background knowledge base, explains a set of examples. The main advantage that ILP has over traditional statistical machine learning approaches is that the learned hypotheses can be easily expressed in plain English and explained to a human user, so facilitating a closer interaction between humans and machines. Traditional ILP frameworks have focused on learning definite logic programs [1–6] and normal logic programs [7,8]. On the other hand, Answer Set Programming [9] is a powerful language for knowledge representation and reasoning. ASP is closely related to other declarative paradigms such as SAT, SMT and Constraint Programming, which have each been used for inductive reasoning [10–12]. Compared with these other paradigms, due to its non-monotonicity, ASP is particularly suited for common-sense reasoning [13–15]. Because of its expressiveness and efficient solving, ASP is

* Corresponding author at: Department of Computing, Huxley Building, 180 Queen's Gate, Imperial College London, London, SW7 2AZ, United Kingdom.
E-mail address: mark.law09@imperial.ac.uk (M. Law).

https://doi.org/10.1016/j.artint.2018.03.005
0004-3702/

also increasingly gaining attention in industry [16]; for example, in decision support systems [17], in e-tourism [18] and in product configuration [19]. Consequently, the scope of ILP has recently been extended to learning answer set programs from examples of partial solutions of a given problem, with the intention being to provide algorithms that support automated learning of complex declarative knowledge. Learning ASP programs allows us to learn a variety of declarative non-monotonic, common-sense theories, including for instance Event Calculus [20] theories [21] and theories for scheduling problems and agents' preference models, both from real user data [22] and from synthetic data [23,24].

Learning ASP programs has several advantages when compared to learning Prolog programs. Firstly, when learning Prolog programs, the goal directed SLDNF procedure of Prolog must be taken into account. Specifically, when learning programs with negation, it must be ensured that the programs are stratified, or otherwise the learned program may loop under certain queries. As ASP is declarative, no such consideration need be taken into account when learning ASP programs. A second, more fundamental advantage of learning ASP programs, is that the theory learned can be expressed using extra types of rules that are not available in Prolog, such as choice rules and weak constraints. Learning choice rules allows us to learn non-deterministic concepts; for instance, we may learn that a coin may non-deterministically land on either *heads* or *tails*, but never both. This could be achieved by learning the simple choice rule `1{heads, tails}1`. Learning choice rules is different from probabilistic ILP settings such as [25–27] where, in similar coins problems the focus would be on learning the probabilities of the two outcomes of are coin. Learning weak constraints enables a natural extension of ILP to preference learning [23], which has resulted to be effective in problem domains such as learning preference models for scheduling [23] and for urban mobility [24].

Several algorithms, aimed at learning under the answer set semantics, and different frameworks for learning ASP programs have been recently introduced in the literature. [28] presented the notions of *brave induction* (ILP_b) and *cautious induction* (ILP_c), based respectively on the well established notions of entailment under the answer set semantics [13,29] of *brave entailment* (when an atom is true in at least one answer set) and *cautious entailment* (when and an atom is true in all answer sets). In brave induction, at least one answer set must cover the examples, whereas in cautious induction, every answer set must cover the examples. Brave induction is actually a special case of an earlier learning framework, called *induction of stable models* (ILP_{sm}) [30], in which examples are partial interpretations. A hypothesis is a solution of an induction of stable models task if for each of the example partial interpretations, there is an answer set of the hypothesis combined with the background knowledge, that covers that partial interpretation. Brave induction is equivalent to induction of stable models with exactly one (partial interpretation) example.

Each of the above frameworks for learning ASP programs is unable to learn some types of ASP programs [31]; for example, brave induction alone cannot learn programs containing hard constraints. In [31], we presented a learning framework, called *Learning from Answer Sets* (ILP_{LAS}), which unifies brave and cautious induction and is able to learn ASP programs containing normal rules, choice rules and hard constraints. In spite of the increased expressivity, none of the above approaches can learn weak constraints, which are able to capture preference learning. Informally, learning weak constraints consists on identifying conditions for ordering answer sets. The learning task in this case would require examples of orderings over partial interpretations. To tackle this aspect of learning ASP programs, we have extended the Learning from Answer Sets framework to Learning from Ordered Answer Sets (ILP_{LOAS}) [23] and demonstrated that our algorithm[1] is able to learn preferences in a scheduling domain. More recently, we have extended the ILP_{LOAS} framework to $ILP^{context}_{LOAS}$, with *context-dependent* examples, which come together with extra contextual information [24].

In this paper, we explore both the expressive power and the computational complexity of each framework. The former is important, as it allows us to identify the class of problems that each framework can solve, whereas the latter gives an indication of the price paid for using each framework. We characterise the expressive power of a framework in terms of new notions called *one-to-one-distinguishability*, *one-to-many-distinguishability* and *many-to-many-distinguishability*. The intuition of one-to-one-distinguishability is that, given some fixed background knowledge B and sufficient examples, the framework should be able to *distinguish* a target hypotheses H_1 from another, unwanted, hypotheses H_2. This means that there should be at least one task T (of the given framework) with background knowledge B, such that H_1 is a solution of T, and H_2 is not. We characterise the one-to-one-distinguishability class of a framework $\mathcal{F}$ (written $\mathcal{D}^1_1(\mathcal{F})$) as the set of tuples $\langle B, H_1, H_2\rangle$ for such B's, H_1's and H_2's, and state that a framework $\mathcal{F}_1$ is more $\mathcal{D}^1_1$-*general* than another $\mathcal{F}_2$ if $\mathcal{F}_2$'s one-to-one-distinguishability class is a strict subset of $\mathcal{F}_1$'s one-to-one-distinguishability class.

One-to-many-distinguishability relates to the task of finding a single target hypothesis from within a set of possible hypotheses. It upgrades the notion of one-to-one-distinguishability classes to one-to-many-distinguishability classes. These are tuples of the form $\langle B, H, S\rangle$ for which a framework has at least one task that includes H and none of the (unwanted) hypotheses in S as an inductive solution. Many-to-many-distinguishability upgrades this notion to many-to-many-distinguishability classes. These contain tuples of the form $\langle B, S_1, S_2\rangle$, where S_1 is a set of target hypotheses, for which a framework must have a task that accepts each hypothesis in S_1 and no hypothesis in S_2 as inductive solution. We show that, under these three measures, $ILP^{context}_{LOAS}$ is more general than ILP_{LOAS}, which is more general than ILP_{LAS}. We also show that ILP_{LAS} is more general than both ILP_{sm} and ILP_c. Although ILP_{sm} is equally $\mathcal{D}^1_1$-general to ILP_b, we show that ILP_{sm} is more general than ILP_b under the one-to-many and many-to-many generality measures.

[1] Our ILASP system for solving ILP_{LOAS} tasks is available for download from [32].

Despite the different generalities of ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$, we show that the computational complexity of all four frameworks is the same, both for the decision problem of verifying that a given hypothesis is a solution of a given learning task, and for the problem of deciding whether a given learning task has any solutions. Similarly, we also show that ILP_{sm} and ILP_b have the same computational complexities for both decision problems, despite the former being more general than the latter under two of our generality measures.

We begin, in Section 2, by reviewing the background material necessary for the rest of the paper. In Section 3 we recall the definitions of each of the learning frameworks and in Sections 4 and 5 we prove the complexities and generalities (respectively) of each learning framework. We conclude the paper with a discussion of the related and future work.

2. Background

2.1. Answer Set Programming

In this section we introduce the concepts needed in the paper. Given any atoms $\mathtt{h}, \mathtt{h}_1, \ldots, \mathtt{h}_k, \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{c}_1, \ldots, \mathtt{c}_m$, $\mathtt{h} :\!\!- \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$ is called a *normal rule*, with $\mathtt{h}$ as the *head* and $\mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$ (collectively) as the *body* ("not" represents negation as failure); a rule $:\!\!- \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$, with an empty head, is a *hard constraint*; a *choice rule* is a rule $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$ (where $\mathtt{l}$ and $\mathtt{u}$ are integers) and its head is called an *aggregate*. A rule R is *safe* if each variable in R occurs in at least one positive literal in the body of R. In this paper we will use $\mathcal{ASP}^{ch}$ to denote the set of *choice programs* P, which are programs composed of safe normal rules, choice rules, and hard constraints. Given a rule R, we will write $head(R)$ to denote the head of R, $body(R)$ to denote the body of R and $body^+$ (resp. $body^-(R)$) to denote the atoms that occur positively (resp. negatively) in the body of R. Given a program P, we will also write $Atoms(P)$ to denote the atoms in P. We will also extend this notation to fragments of a program.

The Herbrand Base of any program $P \in \mathcal{ASP}^{ch}$, denoted HB_P, is the set of variable free (ground) atoms that can be formed from predicates and constants in P. The subsets of HB_P are called the (Herbrand) interpretations of P. A ground aggregate $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_k\}\mathtt{u}$ is satisfied by an interpretation I iff $\mathtt{l} \leq |I \cap \{\mathtt{h}_1, \ldots, \mathtt{h}_k\}| \leq \mathtt{u}$.

As we restrict our programs to sets of normal rules, (hard) constraints and choice rules, we can use the simplified definitions of the *reduct* for choice rules presented in [33]. Given a program P and an Herbrand interpretation $I \subseteq HB_P$, the reduct P^I is constructed from $ground(P)$ (the set of ground instances of rules in P) in 4 steps: firstly, remove rules whose bodies contain the negation of an atom in I; secondly, remove all negative literals from the remaining rules; thirdly, replace the head of any hard constraint, or any choice rule whose head is not satisfied by I with $\bot$ (where $\bot \notin HB_P$); and finally, replace any remaining choice rule $\mathtt{l}\{\mathtt{h}_1, \ldots, \mathtt{h}_m\}\mathtt{u} :\!\!- \mathtt{b}_1, \ldots, \mathtt{b}_n$ with the set of rules $\{\mathtt{h}_i :\!\!- \mathtt{b}_1, \ldots, \mathtt{b}_n \mid \mathtt{h}_i \in I \cap \{\mathtt{h}_1, \ldots, \mathtt{h}_m\}\}$. Any $I \subseteq HB_P$ is an *answer set* of P if it is the minimal model of the *reduct* P^I. Throughout the paper we denote the set of answer sets of a program P with $AS(P)$.

We say a program P *bravely entails* an atom $\mathtt{a}$ (written $P \models_b \mathtt{a}$) if there is at least one answer set A of P such that $\mathtt{a} \in A$. Similarly, P *cautiously entails* $\mathtt{a}$ (written $P \models_c \mathtt{a}$) if for every answer set A of P, $\mathtt{a} \in A$.

Unlike hard constraints in ASP, *weak constraints* do not affect what is, or is not, an answer set of a program P. Hence the above definitions also apply to programs with weak constraints. Weak constraints create an ordering over $AS(P)$ specifying which answer sets are "preferred" to others. A *weak constraint* is of the form $:\sim \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m.[\mathtt{w@l}, \mathtt{t}_1, \ldots, \mathtt{t}_k]$ where $\mathtt{b}_1, \ldots, \mathtt{b}_n$, $\mathtt{c}_1, \ldots, \mathtt{c}_m$ are atoms, $\mathtt{w}$ and $\mathtt{l}$ are terms specifying the *weight* and the *level*, and $\mathtt{t}_1, \ldots, \mathtt{t}_k$ are terms. A weak constraint W is *safe* if every variable in W occurs in at least one positive literal in the body of W. At each *priority level* $\mathtt{l}$, the aim is to discard any answer set which does not minimise the sum of the weights of the ground weak constraints with level $\mathtt{l}$ whose bodies are true. The higher levels are minimised first. The terms $\mathtt{t}_1, \ldots, \mathtt{t}_k$ specify which ground weak constraints should be considered unique [34]. For any program P and an interpretation A, $weak(P, A)$ is the set of tuples $(\mathtt{w}, \mathtt{l}, \mathtt{t}_1, \ldots, \mathtt{t}_k)$ for which there is some $:\sim \mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m.[\mathtt{w@l}, \mathtt{t}_1, \ldots, \mathtt{t}_k]$ in $ground(P)$ such that A satisfies $\mathtt{b}_1, \ldots, \mathtt{b}_n, \mathtt{not}\ \mathtt{c}_1, \ldots, \mathtt{not}\ \mathtt{c}_m$. For each level l, the *score* of the interpretation A is the sum of the weights of tuples with level l, formally $P^l_A = \sum_{(w,l,t_1,\ldots,t_k) \in weak(P,A)} w$. For $A_1, A_2 \in AS(P)$, A_1 *dominates* A_2 (written $A_1 \succ_P A_2$) iff $\exists l$ such that $P^l_{A_1} < P^l_{A_2}$ and $\forall m > l$, $P^m_{A_1} = P^m_{A_2}$. An answer set $A \in AS(P)$ is *optimal* if it is not dominated by any $A_2 \in AS(P)$.

Example 1. Let P be the program $\{0\{\mathtt{p(1)}, \mathtt{p(2)}, \mathtt{p(3)}\}1.\}$. P has 8 answer sets, which are the various combinations of making each of the three $\mathtt{p}$ atoms true or false. Consider the two weak constraints $:\sim \mathtt{p(X)}.[\mathtt{1@1}]$ and $:\sim \mathtt{p(X)}.[\mathtt{1@1, X}]$. The first weak constraint states that if any of the $\mathtt{p}$ atoms is true then a penalty of one must be paid. This penalty is only paid once, regardless whether 1, 2 or 3 of the $\mathtt{p}$ atoms are true. Conversely, the second weak constraint says that a penalty of 1 must be paid for each of the $\mathtt{p}$ atoms that is true. In both cases, $\emptyset$ is the only optimal answer set; however, in the first case, none of the remaining answer sets dominate each other, whereas in the second case, the answer sets with only one $\mathtt{p}$ atom dominate those with 2 $\mathtt{p}$ atoms, which in turn each dominate the single answer set with 3 $\mathtt{p}$ atoms.

Note that the definition of weak constraints used in this paper is in line with the recent ASP standard established in [34]. The syntax of some previous definitions of weak constraints such as [13] do not include the terms $\mathtt{t}_1, \ldots, \mathtt{t}_k$ and considered

every ground instance of every weak constraint individually. This semantics can be achieved using the notion of weak constraints in [34]. Any weak constraint $:\sim$ body.[w : l][2] can be mapped to the weak constraint $:\sim$ body.[w@l, $V_1, \ldots, V_n$], where $V_1, \ldots, V_n$ is the set of all variables that occur in body. If there are multiple weak constraints, to exactly preserve the semantics of [13], a unique term must be added to each weak constraint. For example, {$:\sim$ p(X).[1 : 1]; $:\sim$ q(X).[1 : 1]} would become {$:\sim$ p(X).[1@1, X, 1]; $:\sim$ q(X).[1@1, X, 2]}. With this additional term, $Weak(P, \{p(a), q(a)\})$ (where P is the program containing the two weak constraints) would be equal to $\{(1, 1, a, 1), (1, 1, a, 2)\}$, leading to a score of 2 at level 1; without the additional term, $Weak(P, \{p(a), q(a)\})$ would equal $\{(1, 1, a)\}$, leading to a score of 1 at level 1.

Unless otherwise stated, when we refer to an ASP program in this paper, we mean a program consisting of a finite set of normal rules, choice rules, hard and weak constraints.

We now introduce some extra notation which will be useful in later sections. Given a set of interpretations S, the set $ord(P, S)$ captures the ordering of the interpretations given by the weak constraints in P. It generalises the *dominates* relation; so it not only includes $\langle A_1, A_2, < \rangle$ if $A_1 \succ_P A_2$, but it also includes tuples for other binary comparison operators. Formally, $\langle A_1, A_2, < \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_1 \succ_P A_2$; $\langle A_1, A_2, > \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_2 \succ_P A_1$; $\langle A_1, A_2, \leq \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_2 \not\succ_P A_1$; $\langle A_1, A_2, \geq \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_1 \not\succ_P A_2$; $\langle A_1, A_2, = \rangle \in ord(P, S)$ if $A_1, A_2 \in S$, $A_1 \not\succ_P A_2$ and $A_2 \not\succ_P A_1$; $\langle A_1, A_2, \neq \rangle \in ord(P, S)$ if $A_1, A_2 \in S$ and $A_1 \succ_P A_2$ or $A_2 \succ_P A_1$. Given an ASP program, we write $ord(P)$ as a shorthand for $ord(P, AS(P))$. Two ASP programs P and Q are *strongly equivalent* (written $P \equiv_s Q$) if for every ASP program R, $AS(P \cup R) = AS(Q \cup R)$.

We now recall the splitting set theorem from [35], which we use in the proofs throughout the paper. This theorem relies on the notions of a splitting set and the partial evaluation of a logic program. Given a program P, a set $U \subseteq HB_P$ is a splitting set of P if and only if for every rule $R \in ground(P)$ such that $Atoms(head(R)) \cap U \neq \emptyset$, $Atoms(R) \subseteq U$. Given a ground rule R and a set of atoms U, we write $R \backslash U$ to denote the rule R with all (positive or negative) occurrences of atoms in U removed from the body of R. Given a program P a splitting set U of P and a set $X \subseteq U$, the partial evaluation of P with respect to U and X, written $e_U(P, X)$, is the program $\{R \backslash U \mid R \in ground(P), Atoms(head(R)) \cap U = \emptyset, (body^+(R) \cap U) \subseteq X, body^-(R) \cap X = \emptyset\}$.

Theorem 1. *Given any ground ASP program P, and splitting set U of P, $AS(P) = \{X \cup Y \mid X \in AS(\{R \in P \mid Atoms(head(R)) \cap U \neq \emptyset\}), Y \in AS(e_U(P, X))\}$.*

The intuition behind the splitting set theorem is that if a set of atoms U is known to *split* the program P, then we can find the answer sets of the subprogram that defines the atoms in U first. For each of these answer sets X, we can partially evaluate P using X and solve this partially evaluated program for answer sets. The splitting set theorem then guarantees that for each answer set Y of the partially evaluated program, $X \cup Y$ is an answer set of P. Furthermore, every answer set of P can be constructed in this way.

2.2. Complexity theory

We assume the reader is familiar with the fundamental concepts of complexity, such as Turing machines and reductions; for a detailed explanation, see [36].

Many of the decision problems for ASP are known to be complete for classes in the polynomial hierarchy [37]. The classes of the polynomial hierarchy are defined as follows: P is the class of all problems which can be solved in polynomial time by a Deterministic Turing Machine (DTM); $\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$; $\Delta_{k+1}^P = P^{\Sigma_k^P}$ is the class of all problems which can be solved by a DTM in polynomial time with a Σ_k^P oracle; $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$ is the class of all problems which can be solved by a non-deterministic Turing Machine in polynomial time with a Σ_k^P oracle; finally, $\Pi_{k+1}^P = \text{co-}NP^{\Sigma_k^P}$ is the class of all problems whose complement can be solved by a non-deterministic Turing Machine in polynomial time with a Σ_k^P oracle. Σ_1^P and Π_1^P are NP and co-NP (respectively), where NP is the class of problems which can be solved by a non-deterministic Turing machine in polynomial time and co-NP is the class of problems whose complement is an NP problem.

DP is the class of problems D that can be mapped to a pair of problems D_1 and D_2 such that $D_1 \in NP$, $D_2 \in$ co-NP, and for each instance I of D, I answers "yes" if and only if both of the mapped instances I_1 and I_2 (of D_1 and D_2, respectively) answer "yes". It is well known [36] that the following inclusions hold: $P \subseteq NP \subseteq DP \subseteq \Delta_2^P \subseteq \Sigma_2^P$ and $P \subseteq$ co-$NP \subseteq DP \subseteq \Delta_2^P \subseteq \Pi_2^P$.

3. Learning frameworks

In this section, we give the definitions of the six learning frameworks we analyse in this paper. The first three – brave induction, cautious induction and induction of stable models – are not our own. We reformulate, but preserve the meaning of, the original definitions for easier comparison with our own.

[2] In [13] ":" was used for the same purpose as "@".

It is common in ILP for a task to have a *hypothesis space* (the set of all rules which can appear in hypotheses). The purpose of the hypothesis space is two-fold: firstly, it allows the task to be restricted to those solutions which are in some way interesting; secondly, it aids the computational search for inductive solutions. Tasks for brave and cautious induction and for induction of stable models were originally presented with no hypothesis space [28,30] as they were mainly considered theoretically without the specifications of efficient algorithmic computations. The only publicly available algorithms for brave induction [38,39] make use of a hypothesis space defined by mode declarations [40]. In this paper, we "upgrade" each of brave induction, cautious induction and induction of stable models with a hypothesis space S_M.

3.1. Notation and terminology

An ILP *learning framework* $\mathcal{F}$ defines what a *learning task* of $\mathcal{F}$ is and what an *inductive solution* is for a given learning task of $\mathcal{F}$. For each framework a task is a tuple $\langle B, S_M, E\rangle$, where B is an ASP program called the *background knowledge*, S_M is a set of ASP rules called the *hypothesis space*, and E is a tuple called the examples. The structure of E depends on the type of ILP framework. Each of the papers [28], [30], [31] and [23] presented learning frameworks with different languages for B and S_M; for example, induction of stable models was presented only for normal logic programs. It would be unfair to say that induction from stable models is not general enough to learn programs with choice rules, simply because they were not considered in the original paper (in fact, induction from stable models *is* general enough to learn some programs with choice rules). For a fair comparison we therefore assume in this paper that every learning framework has a background knowledge B and hypothesis space S_M that consist of normal rules, choice rules, hard constraints and weak constraints.

Given a framework $\mathcal{F}$ and a learning task $T_\mathcal{F} = \langle B, S_M, E\rangle$ of $\mathcal{F}$, a *hypothesis* is any subset of the hypothesis space S_M. In Section 5, we consider tasks with unrestricted hypothesis spaces (written $\langle B, E\rangle$), in which case any ASP program can be called a hypothesis. An *inductive solution* is a hypothesis that, together with the background knowledge B, satisfies some conditions on E (given by the particular learning framework $\mathcal{F}$). We write $ILP_\mathcal{F}(T_\mathcal{F})$ to denote the set of all inductive solutions of $T_\mathcal{F}$. Throughout the paper, we use the term *covers* to apply to any kind of example: i.e. given a $\mathcal{F}$ task $\langle B, S_M, E\rangle$, we say that a hypothesis H covers an example e (any element of any component of E), if it meets the particular conditions that the framework $\mathcal{F}$ puts on H and e.

3.2. Framework definitions

Brave induction (ILP_b), first presented in [28], defines an inductive task in which all examples are ground atoms that should be covered in at least one answer set, i.e. entailed under brave entailment in ASP. The original definition did not consider atoms which should not be present in an answer set, namely negative examples. The two publicly available algorithms that realise brave induction, on the other hand, do allow for negative examples. We therefore upgrade the definition in this paper to allow negative examples[3] as follows.

Definition 1. A brave induction (ILP_b) task T_b is a tuple $\langle B, S_M, \langle E^+, E^-\rangle\rangle$, where B is an ASP program called the background knowledge, S_M is the hypothesis space and E^+ and E^- are sets of ground atoms called the positive and negative examples (respectively). A hypothesis $H \subseteq S_M$ is said to be an inductive solution of T_b (written $H \in ILP_b(T_b)$) if and only if $\exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

Cautious induction (ILP_c) was also first presented in [28]. It defines an inductive task where all of the examples should be covered in every answer set (i.e. entailed under cautious entailment in ASP) and that $B \cup H$ should be satisfiable (have at least one answer set). Similarly to brave induction, the original definition did not consider negative examples, but in Definition 2 we upgrade the framework to include negative examples.

Definition 2. A cautious induction (ILP_c) task T_c is a tuple $\langle B, S_M, \langle E^+, E^-\rangle\rangle$, where B is an ASP program called the background knowledge, S_M is the hypothesis space and E^+ and E^- are sets of ground atoms called the positive and negative examples (respectively). A hypothesis $H \subseteq S_M$ is said to be an inductive solution of T_c (written $H \in ILP_c(T_c)$) if and only if $AS(B \cup H) \neq \emptyset$ and $\forall A \in AS(B \cup H)$, $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.

Brave induction alone can only reason about what should be true (or false) in a single answer set of $B \cup H$. It cannot specify other brave tasks such as enforcing that two atoms are both bravely entailed, but not necessarily in the same answer set. *Induction of stable models* [30] (ILP_{sm}), on the other hand, generalises the notion of brave induction as shown in Definition 4. The following terminology is first introduced.

Definition 3. A *partial interpretation* e is a pair of sets of ground atoms $\langle e^{inc}, e^{exc}\rangle$. An interpretation I is said to *extend* e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.

[3] Note that in ILP_b a negative example `e_i` can be easily simulated by adding a rule `a_i :- not e_i` to the background knowledge and giving `a_i` as a positive example (where `a_i` is a new atom (unique to `e_i`) which does not appear anywhere in the original task).

Definition 4. An induction of stable models (ILP_{sm}) task T_{sm} is a tuple $\langle B, S_M, \langle E \rangle\rangle$, where B is an ASP program called the background knowledge, S_M is the hypothesis space and E is a set of partial interpretations called the examples. A hypothesis H is said to be an inductive solution of T_{sm} (written $H \in ILP_{sm}(T_{sm})$) if and only if $H \subseteq S_M$ and $\forall e \in E$, $\exists A \in AS(B \cup H)$ such that A extends e.

Note that a brave induction task can be thought of as a special case of induction of stable models, with exactly one (partial interpretation) example.

We now consider the *Learning from Answer Sets* framework introduced in [31]. This is the first framework capable of unifying the concepts of brave and cautious induction. The idea is to use examples of partial interpretations which should or should not be extended by answer sets of $B \cup H$.

Definition 5. A *Learning from Answer Sets* task is a tuple $T = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ where B is an ASP program called the background knowledge, S_M is the hypothesis space and E^+ and E^- are sets of partial interpretations called, respectively, the positive and negative examples. A hypothesis $H \subseteq S_M$ is an *inductive solution* of T (written $H \in ILP_{LAS}(T)$) if and only if:

1. $\forall e^+ \in E^+ \ \exists A \in AS(B \cup H)$ such that A extends e^+
2. $\forall e^- \in E^- \ \nexists A \in AS(B \cup H)$ such that A extends e^-

Note that this definition combines properties of both the brave and cautious semantics: the positive examples must each be bravely entailed, whereas the negation of each negative example must be cautiously entailed.

Example 2. Consider an ILP_{LAS} learning task whose background knowledge B contains definitions of the structure of a `4x4` Sudoku board; i.e. definitions of `cell`, `same_row`, `same_col` and `same_block` (where `same_row`, `same_col` and `same_block` are true only for two *different* cells in the same row, column or block).

$$B = \left\{ \begin{array}{l} \texttt{cell((1,1)).}\ \ \texttt{cell((1,2)).}\ \ \ldots\ \ \texttt{cell((4,4)).} \\ \texttt{same\_row((X1,Y),(X2,Y)):-cell((X1,Y)),cell((X2,Y)),X1} \neq \texttt{X2.} \\ \texttt{same\_col((X,Y1),(X,Y2)):-cell((X,Y1)),cell((X,Y2)),Y1} \neq \texttt{Y2.} \\ \texttt{block((1,1),1).}\ \ \texttt{block((1,2),1).}\ \ \texttt{block((2,1),1).}\ \ \texttt{block((2,2),1).} \\ \texttt{block((3,1),2).}\ \ \texttt{block((3,2),2).}\ \ \texttt{block((4,1),2).}\ \ \texttt{block((4,2),2).} \\ \texttt{block((1,3),3).}\ \ \texttt{block((1,4),3).}\ \ \texttt{block((2,3),3).}\ \ \texttt{block((2,4),3).} \\ \texttt{block((3,3),4).}\ \ \texttt{block((3,4),4).}\ \ \texttt{block((4,3),4).}\ \ \texttt{block((4,4),4).} \\ \texttt{same\_block(C1,C2):-block(C1,B),block(C2,B),C1} \neq \texttt{C2.} \end{array} \right\}$$

For the purposes of this example, we will consider only a small hypothesis space S_M but in practice this would be much larger.[4]

$$S_M = \left\{ \begin{array}{l} \texttt{0\{value(C,1),value(C,2),value(C,3),value(C,4)\}1:-cell(C).} \\ \texttt{1\{value(C,1),value(C,2),value(C,3),value(C,4)\}1:-cell(C).} \\ \texttt{1\{value(C,1),value(C,2),value(C,3),value(C,4)\}2:-cell(C).} \\ \texttt{:-same\_row(C1,C2),value(C1,V),value(C2,V).} \\ \texttt{:-same\_col(C1,C2),value(C1,V),value(C2,V).} \\ \texttt{:-same\_block(C1,C2),value(C1,V),value(C2,V).} \end{array} \right\}$$

$$E^+ = \left\{ \langle \{\texttt{value((1,1),1)}\}, \emptyset \rangle \right\}$$

$$E^- = \left\{ \begin{array}{l} \langle \{\texttt{value((1,1),1)}, \texttt{value((1,3),1)}\}, \emptyset \rangle \\ \langle \{\texttt{value((1,1),1)}, \texttt{value((3,1),1)}\}, \emptyset \rangle \\ \langle \{\texttt{value((1,1),1)}, \texttt{value((2,2),1)}\}, \emptyset \rangle \\ \langle \{\texttt{value((1,1),1)}, \texttt{value((1,1),2)}\}, \emptyset \rangle \\ \langle \emptyset, \{\texttt{value((1,1),1)}, \texttt{value((1,1),2)}, \texttt{value((1,1),3)}, \texttt{value((1,1),4)}\} \rangle \end{array} \right\}$$

We need to be able to say that there should be at least one answer set that assigns a value to a cell, or otherwise the empty hypothesis would be sufficient. This is captured by our positive example which causes at least one of the choice rules to be part of a solution in order to be covered. Our first three negative examples require the three constraints to be also included in a solution. Without each one of these negative examples, at least one constraint could be left out of the solution. The fourth negative example means that the upper bound of the counting aggregate in the choice rule must be 1, as otherwise there would be answer sets in which cell `(1,1)` was assigned to both `1` and `2`. Finally, the fifth negative

[4] A larger version of this learning task with a hypothesis space with 542 rules can be found in the manual for our learning algorithm, ILASP [41].

example forces that the lower bound of the choice rule should be `1` as otherwise there would be answer sets in which `(1, 1)` was not assigned to any of the values between `1` and `4`. Hence, one possible inductive solution is:

$$H = \left\{ \begin{array}{l} \texttt{1\{value(C, 1), value(C, 2), value(C, 3), value(C, 4)\}1 :- cell(C).} \\ \texttt{:- same\_row(C1, C2), value(C1, V), value(C2, V).} \\ \texttt{:- same\_col(C1, C2), value(C1, V), value(C2, V).} \\ \texttt{:- same\_block(C1, C2), value(C1, V), value(C2, V).} \end{array} \right\}$$

The only other solutions within the hypothesis space S_M are those that contain H and also extra redundant choice rules, such as `0{value(C, 1), value(C, 2), value(C, 3), value(C, 4)}1 :- cell(C).`

Note that we need ILP_{LAS}'s combination of brave and cautious induction to separate the correct hypothesis from the incorrect hypotheses.

- If we instead use brave induction, whichever examples we use, if H is a solution, then any of the choice rules on their own is also a solution. For instance, consider the hypothesis H', containing only the choice rule `0{value(C, 1), value(C, 2), value(C, 3), value(C, 4)}1 :- cell(C).` For any examples $\langle E^+, E^- \rangle$ such that $H \in ILP_b(\langle B, \langle E^+, E^- \rangle\rangle)$, there must be an answer set A of $B \cup H$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$. As $AS(B \cup H) \subset AS(B \cup H')$, any such answer set is also an answer set of $B \cup H'$; and hence, H' is also a solution of the task.
- If we use cautious induction, we have to give examples which are either true in every answer set, or false in every answer set. Therefore, we could not give any examples about the `value` predicate – for each atom `value(x, y)` (where `x` and `y` range from 1 to 4), there is at least one answer set of $B \cup H$ that contains `value(x, y)` and at least one that does not; this means that if `value(x, y)` is given as either a positive or negative example, H will not be a solution of the task.
  This means that for any ILP_c task T_c such that H is a solution, any subset of the hypothesis space S_M is also a solution of T_c.

Note that none of the learning frameworks we have considered so far (ILP_{LAS} included) can incentivise learning a weak constraint. This is because the frameworks only have examples of what should be in some, all or none of the answer sets of $B \cup H$. Any solution H containing a weak constraint W will have the same answer sets with W removed and $H \backslash \{W\}$ would therefore be a shorter (more optimal[5]) solution. The notion of *ordering examples* is needed to incentivise learning weak constraints, in order to enforce which answer sets of $B \cup H$ should dominate other answer sets.

Definition 6. An *ordering example* is a tuple $o = \langle e_1, e_2, op \rangle$ where e_1 and e_2 are partial interpretations and op is a binary comparison operator ($<$, $>$, $=$, $\leq$, $\geq$ or $\neq$). An ASP program P *bravely respects* o iff $\exists A_1, A_2 \in AS(P)$ such that all of the following conditions hold: (i) A_1 extends e_1; (ii) A_2 extends e_2; and (iii) $\langle A_1, A_2, op \rangle \in ord(P)$. P *cautiously respects* o iff $\nexists A_1, A_2 \in AS(P)$ such that all of the following conditions hold: (i) A_1 extends e_1; (ii) A_2 extends e_2; and (iii) $\langle A_1, A_2, op \rangle \notin ord(P)$.

Note that Definition 6 generalises our initial definition of ordering examples given in [23], where ordering examples had only the operator $<$, and we could not express examples of pairs of answer sets which were equally preferred. In Section 5 we show that this extension allows us to learn a wider class of programs. We now define the notion of *Learning from Ordered Answer Sets* (ILP_{LOAS}).

Definition 7. A *Learning from Ordered Answer Sets* task is a tuple $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle\rangle$ where B is an ASP program, called the background knowledge, S_M is the hypothesis space, E^+ and E^- are sets of partial interpretations called, respectively, positive and negative examples, and O^b and O^c are sets of ordering examples over E^+ called brave and cautious orderings. A hypothesis $H \subseteq S_M$ is an inductive solution of T (written $H \in ILP_{LOAS}(T)$) if and only if:

1. $H \in ILP_{LAS}(\langle B, S_M, \langle E^+, E^- \rangle\rangle)$
2. $\forall o \in O^b$ $B \cup H$ bravely respects o
3. $\forall o \in O^c$ $B \cup H$ cautiously respects o

Note that the orderings are only over positive examples. We chose to make this restriction as there does not appear to be any scenario where a hypothesis would need to respect orderings which are not extended by any pair of answer sets of $B \cup H$.

Example 3. Consider the ILP_{LOAS} task $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle\rangle$ where the individual components of the task are as follows:

[5] It is common practice in ILP to search for the optimal (shortest) solution(s).

- $B = \{0\{\mathtt{p}, \mathtt{q}\}2\,.\}$
- S_M is unrestricted (i.e. S_M is the set of all normal rules, choice rules and hard and weak constraints).
- $E^+ = \{e_1^+, e_2^+\}$ where $e_1^+ = \langle\{\mathtt{p}\}, \emptyset\rangle$ and $e_2^+ = \langle\emptyset, \{\mathtt{p}\}\rangle$
- $E^- = \emptyset$
- $O^b = \{\langle e_1^+, e_2^+, <\rangle\}$
- $O^c = \{\langle e_1^+, e_1^+, =\rangle\}$

The positive examples of this task are already satisfied by the background knowledge, which has the answer sets $\emptyset$, {p}, {q} and {p, q}. As there are no negative examples, it remains to find a set of weak constraints such that there is at least one answer set which contains p which is preferred to at least one answer set which does not contain p and all answer sets which contain p are equally optimal.

One such hypothesis is the single weak constraint `:~ not p.[1@1]`.

The frameworks discussed so far have examples which can only express the properties of a learned hypothesis H together with a fixed background knowledge B. These properties are on the answer sets of $B \cup H$ (and the ordering of these answer sets). In [24], we presented a new learning framework that uses *context-dependent* examples. Each example comes with its own *context*, which is an $\mathcal{ASP}^{ch}$ program C. Examples then express properties of $B \cup H \cup C$, meaning that by using multiple examples (with different contexts), we can express that $B \cup H \cup C_1$ should have some properties and that $B \cup H \cup C_2$ should have different properties.

Definition 8. A *context-dependent partial interpretation* (CDPI) is a pair $\langle e, C\rangle$, where e is a partial interpretation and C is an $\mathcal{ASP}^{ch}$ program, called a *context*. A *context-dependent ordering example* (CDOE) o is a tuple $\langle\langle e_1, C_1\rangle, \langle e_2, C_2\rangle, op\rangle$, where the first two elements are CDPIs and op is a binary comparison operator ($<$, $>$, $=$, $\leq$, $\geq$ or $\neq$). P is said to *bravely respect* o if $\exists A_1 \in AS(P \cup C_1), \exists A_2 \in AS(P \cup C_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \in ord(P, AS(P \cup C_1) \cup AS(P \cup C_2))$. A program P is said to *cautiously respect* o if $\forall A_1 \in AS(P \cup C_1), \forall A_2 \in AS(P \cup C_2)$ such that A_1 extends e_1 and A_2 extends e_2, $\langle A_1, A_2, op\rangle \in ord(P, AS(P \cup C_1) \cup AS(P \cup C_2))$.

When examples are given with empty contexts, they are equivalent to examples in ILP_{LOAS}. Note also that contexts do not contain weak constraints. In fact, the operator $\succ_P$ defines the ordering over two answer sets based on the weak constraints in one program P. So, given a CDOE $\langle\langle e_1, C_1\rangle, \langle e_2, C_2\rangle\rangle$ such that C_1 and C_2 contain different weak constraints, it is not clear which program to consider for computing the ordering of answer sets – i.e. whether they should be checked against the weak constraints in P, $P \cup C_1$, $P \cup C_2$ or $P \cup C_1 \cup C_2$.

We now present a formal definition of the $ILP^{context}_{LOAS}$ framework.

Definition 9. A *Context-dependent Learning from Ordered Answer Sets* ($ILP^{context}_{LOAS}$) task is a tuple $T = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ where B is an ASP program called the background knowledge, S_M is the set of rules allowed in the hypotheses (the hypothesis space), E^+ and E^- are finite sets of CDPIs called, respectively, positive and negative examples, and O^b and O^c are finite sets of CDOEs over E^+ called, respectively, brave and cautious context-dependent orderings. A hypothesis $H \subseteq S_M$ is an inductive solution of T (written $H \in ILP^{context}_{LOAS}(T)$) if and only if:

1. $\forall\langle e^+, C\rangle \in E^+, \exists A \in AS(B \cup C \cup H)$ st A extends e^+
2. $\forall\langle e^-, C\rangle \in E^-, \nexists A \in AS(B \cup C \cup H)$ st A extends e^-
3. $\forall o \in O^b$, $B \cup H$ bravely respects o
4. $\forall o \in O^c$, $B \cup H$ cautiously respects o

In [24], we showed that context-dependent examples could be used to simplify the encoding of certain tasks, by splitting the background knowledge into contexts that were only relevant to particular examples. Although any $ILP^{context}_{LOAS}$ task can be transformed into an ILP_{LOAS} task, in general this requires parts of the examples to be encoded in the background knowledge. Example 4 shows such a transformation.

Example 4. Consider a simple scenario where we have a machine that has a single configuration parameter a, which is allowed to take any natural number as its value. A user is allowed to input another natural number b, and if $a > b$, the machine should beep.

Two example scenarios could be encoded as the context-dependent positive examples $\langle\{\mathtt{beep}\}, \emptyset\rangle, \{\mathtt{value(a,3).\ value(b,2).}\}\rangle$, and $\langle\langle\emptyset, \{\mathtt{beep}\}\rangle, \{\mathtt{value(a,4).\ value(b,20).}\}\rangle$. A task containing these examples and an empty background knowledge requires an inductive solution that when combined with the context of the first example would have at least one answer set containing `beep`, and when combined with the second example would have at least one answer set not containing `beep`. If we were expressing the same task in ILP_{LOAS} the above two scenarios would be represented considering the background knowledge:

Table 1
A summary of the available systems for learning under the answer set semantics.

Framework	Systems
ILP_b	XHAIL [42], ASPAL [38] and RASPAL [43]
ILP_{sm}	
ILP_c	
ILP_{LAS}	ILASP [32]
ILP_{LOAS}	ILASP [32]
$ILP^{context}_{LOAS}$	ILASP [32]

Table 2
A summary of the complexity of the various learning frameworks.

Framework	Complexity of verification	Complexity of deciding satisfiability
ILP_b	NP-complete	NP-complete
ILP_{sm}	NP-complete	NP-complete
ILP_c	DP-complete	Σ^p_2-complete
ILP_{LAS}	DP-complete	Σ^p_2-complete
ILP_{LOAS}	DP-complete	Σ^p_2-complete
$ILP^{context}_{LOAS}$	DP-complete	Σ^p_2-complete

$$B = \{\ \texttt{1\{value(a, 3), value(a, 4)\}1 . 1\{value(b, 2), value(b, 20)\}1 .}\ \}$$

The context-dependent examples could then be mapped to the non context-dependent examples $\langle\{\texttt{value(a, 3)}, \texttt{value(b, 2)}, \texttt{beep}\}, \emptyset\rangle$ and $\langle\{\texttt{value(a, 4)}, \texttt{value(b, 20)}\}, \{\texttt{beep}\}\rangle$. In fact, in [24], we show that there is a general mapping from $ILP^{context}_{LOAS}$ to ILP_{LOAS}. This mapping, just as the simplified mapping here, depends on encoding the examples in the background knowledge, which abuses the purpose of the background knowledge. The contexts in context-dependent examples allow us instead to separate information that is truly background knowledge, which applies in all scenarios, from information that is part of a particular example.

3.3. Systems for learning under the answer set semantics

The current publicly available systems for ILP can be categorised according to the 6 frameworks presented in this section (Table 1). It should be noted that although there are no systems which directly solve ILP_c or ILP_{sm} tasks, both can be simply translated into ILP_{LAS} tasks, and can therefore be solved by the ILASP system.

The ILED [21] system is an incremental extension of XHAIL, that is specifically targeted at learning Event Calculus [20] theories. The underlying mechanism is based on brave induction, but each of its examples are in terms of two sequential time points.

4. Complexity

In this section, we discuss the complexity of each of the learning frameworks presented in Section 3 with respect to two decision problems: *verification*, deciding whether a given hypothesis H is an inductive solution of a task T; and *satisfiability*, deciding whether a learning task T has any inductive solutions. A summary of the results is shown in Table 2. To aid readability, the proofs of the propositions stated in this section are given in appendix. All complexities discussed in this section are for propositional versions of the frameworks (both the background knowledge and hypothesis space of each learning task is ground).

4.1. Learning from answer sets with stratified summing aggregates

As there are existing results on the complexity of solving *aggregate stratified* programs, it is useful to introduce a new learning framework ILP^s_{LAS}, which is a generalization of ILP_{LAS}, that allows *summing aggregates* in the bodies of rules, as long as they are *stratified*. The existing results on the complexity of these programs then allow us to prove the complexity of ILP^s_{LAS}. Hence, as we can show that ILP_{LOAS} reduces to ILP^s_{LAS}, this is helpful in proving the complexity of ILP_{LOAS}.

A summing aggregate s is of the form $\texttt{l\#sum\{a}_1 = \texttt{w}_1, \ldots, \texttt{a}_\texttt{n} = \texttt{w}_\texttt{n}\}\texttt{u}$, where $\texttt{l}$, $\texttt{u}$ and $\texttt{w}_1, \ldots, \texttt{w}_\texttt{n}$ are integers and $\texttt{a}_1, \ldots, \texttt{a}_\texttt{n}$ are atoms. s is satisfied by an interpretation I if and only if $\texttt{l} \leq \left(\sum_{\texttt{w}_\texttt{i} \in WS} w_i\right) \leq \texttt{u}$, where WS is the set $\{\texttt{w}_\texttt{i} \mid i \in [0..n], \texttt{a}_\texttt{i} \in I\}$. We now recall the definition of *aggregate stratification* from [44]. We slightly simplify the definition by considering only propositional programs without disjunction.

Definition 10. A propositional logic program P, in which aggregates occur only in bodies of rules, is *stratified on an aggregate* $\texttt{agg}$ if there is a level mapping $\|\ \|$ from $Atoms(P)$ to ordinals, such that for each rule $R \in P$, the following holds:

1. $\forall \mathtt{b} \in Atoms(body(R)) : ||\mathtt{b}|| \leq ||head(R)||$
2. If $\mathtt{agg} \in body(R)$, then $\forall \mathtt{b} \in Atoms(\mathtt{agg}) : ||\mathtt{b}|| < ||head(R)||$

P is said to be *aggregate stratified* if it is stratified on every aggregate in P.

The intuition is that aggregate stratification forbids recursion through aggregates. In general aggregate stratified programs have a lower complexity than non-aggregate stratified programs. Aggregate stratification has nothing to do with negation as failure, and therefore, whether a program is aggregate stratified is unrelated to whether it is *stratified* in the usual sense. Note that constraints and choice rules can be added in to any aggregate stratified program without breaking stratification so long as no atoms in the head of the choice rule are on a lower level than any atom in the body. This is illustrated by the following example.

Example 5. Any constraint $\mathtt{:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m}$ can be rewritten as $\mathtt{s:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,not\ s}$ where $\mathtt{s}$ is a new atom. $\mathtt{s}$ can then be mapped to a higher level than any other atom.

A choice rule $\mathtt{l\{h_1,\ldots,h_o\}u:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m}$ can be rewritten as:

$$\mathtt{h_1:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,not\ h'_1.}$$
$$\mathtt{h'_1:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,not\ h_1.}$$
$$\ldots$$
$$\mathtt{h_o:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,not\ h'_o.}$$
$$\mathtt{h'_o:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,not\ h_o.}$$

$$\mathtt{s:-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,\{h_1,\ldots,h_n\}l-1,not\ s.}$$
$$\mathtt{s':-\ b_1,\ldots,b_n,not\ c_1,\ldots,not\ c_m,u+1\{h_1,\ldots,h_n\},not\ s'.}$$

where $\mathtt{h'_1},\ldots,\mathtt{h'_o},\mathtt{s},\mathtt{s'}$ are all new atoms. $\mathtt{s}$ and $\mathtt{s'}$ can both be given a new highest level and each $\mathtt{h'_i}$ can be given the same level as $\mathtt{h_i}$ (if they did not occur in the previous program then they should be given a new level one below $\mathtt{s}$ and $\mathtt{s'}$). Provided the previous program was aggregate stratified, then this new one is too. To avoid constantly using this mapping, we will refer to programs with choice rules and constraints as also being aggregate stratified.

Lemma 1. *[44] Deciding whether an aggregate stratified propositional program without disjunction cautiously entails an atom is co-NP-complete.*

Corollary 1. *Deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is NP-complete.*

Proof. We first show that deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is in NP. We do this by showing that there is a polynomial reduction from this problem to the complement of the problem in Lemma 1 (which by definition of co-NP must be in NP). The complement of the problem in Lemma 1 is deciding whether a non disjunctive aggregate stratified program does not cautiously entail an atom. Take any non-disjunctive aggregate stratified program P and any atom $\mathtt{a}$ and let $\mathtt{neg\_a}$ be an atom that does not occur in P. $P \models_b \mathtt{a}$ if and only if $P \cup \{\mathtt{neg\_a:-\ not\ a.}\} \not\models_c \mathtt{neg\_a}$. So the decision problem is in NP.

It remains to show that deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom is NP-hard. We do this by showing that any problem in NP can be reduced in polynomial time to deciding the satisfiability of an aggregate stratified propositional program without disjunction.

Consider an arbitrary NP problem D. The complement of D, $\bar{D}$, must be in co-NP (by definition of co-NP). Hence, by Lemma 1, there is a polynomial reduction from $\bar{D}$ to deciding whether an aggregate stratified propositional program without disjunction cautiously entails an atom. We define the polynomial reduction from D to deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom as follows: for any instance I of D, let P and $\mathtt{a}$ be the program and atom given by the polynomial reduction from the complement of I to deciding cautious entailment; define P' as the program $P \cup \{\mathtt{neg\_a:-\ not\ a.}\}$ (where $\mathtt{neg\_a}$ is a new atom). I returns true if and only if $P \not\models_c \mathtt{a}$ if and only if $P' \models_b \mathtt{neg\_a}$. Hence, as P' is still aggregate stratified (the new atom $\mathtt{neg\_a}$ can be put in the top strata), this is a polynomial reduction from D to deciding whether an aggregate stratified propositional program without disjunction bravely entails an atom. Hence, the decision problem is NP-hard. □

We can now introduce our extra learning task, *Learning from Answer Sets with Stratified Aggregates* (ILP^s_{LAS}). It is the same as Learning from Answer Sets, except for allowing summing aggregates in the bodies of the rules in B and S_M, as long as

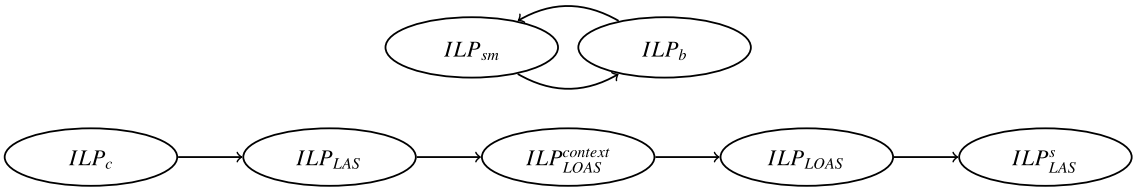

Fig. 1. Chains of polynomial reductions where each arrow denotes that there is a polynomial reduction from one framework to another.

$B \cup S_M$ is aggregate stratified. Note that the condition of $B \cup S_M$ being aggregate stratified implies that for any hypothesis $H \subseteq S_M$, $B \cup H$ is aggregate stratified.

4.2. Relationships between the learning tasks

In this section we prove for both decision problems that ILP_b and ILP_{sm} both reduce to each other polynomially. We also show that for both decision problems there is a chain of polynomial reductions from ILP_c to ILP_{LAS} to $ILP^{context}_{LOAS}$ to ILP_{LOAS} to ILP^{s}_{LAS}. This chain of reductions is then used in proving that all four tasks share the same complexity for both decision problems. By proving that ILP_c is $\mathcal{O}$-hard and ILP^{s}_{LAS} is in $\mathcal{O}$ for some complexity class $\mathcal{O}$, we prove that all four tasks are $\mathcal{O}$-complete. Similarly as ILP_b and ILP_{sm} both reduce polynomially to each other for both decision problems, if for one of the problems ILP_b is $\mathcal{O}$-complete for some class then so is ILP_{sm}. The chains of reductions are shown in Fig. 1.

Proposition 1 shows that the complexity of ILP_b and ILP_{sm} coincide for both decision problems.

Proposition 1.

1. *Deciding both verification and satisfiability for ILP_b reduces polynomially to the corresponding ILP_{sm} decision problem.*
2. *Deciding both verification and satisfiability for ILP_{sm} reduces polynomially to the corresponding ILP_b decision problem.*

Proposition 2 shows that there is a chain of polynomial reductions from ILP_c to ILP_{LAS} to ILP_{LOAS} to $ILP^{context}_{LOAS}$ to ILP^{s}_{LAS} for both decision problems.

Proposition 2.

1. *Deciding both verification and satisfiability for ILP_c reduces polynomially to the corresponding ILP_{LAS} decision problem.*
2. *Deciding both verification and satisfiability for ILP_{LAS} reduces polynomially to the corresponding $ILP^{context}_{LOAS}$ decision problem.*
3. *Deciding both verification and satisfiability for $ILP^{context}_{LOAS}$ reduces polynomially to the corresponding ILP_{LOAS} decision problem.*
4. *Deciding both verification and satisfiability for ILP_{LOAS} reduces polynomially to the corresponding ILP^{s}_{LAS} decision problem.*

4.3. Complexity of deciding verification and satisfiability for each framework

For each of the learning frameworks, we prove the complexity of deciding verification and satisfiability. We start with the ILP_b and ILP_{sm} frameworks, for which both decision problems are NP-complete.

Proposition 3. *Verifying whether a given H is an inductive solution of a general ILP_b task is NP-complete.*

Corollary 2. *Verifying whether a given H is an inductive solution of a general ILP_{sm} task is NP-complete.*

Proposition 4. *Deciding the satisfiability of a general ILP_b task is NP-complete.*

Corollary 3. *Deciding the satisfiability of a general ILP_{sm} task is NP-complete.*

We have now proven the complexity of deciding verification and satisfiability for ILP_b and ILP_{sm}, proving the corresponding entries in Table 2. It remains to show the complexities for ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.

As we have shown that ILP_c reduces to ILP_{LAS} which, in turn, reduces to ILP_{LOAS}, which reduces to $ILP^{context}_{LOAS}$ and that $ILP^{context}_{LOAS}$ reduces to ILP^{s}_{LAS} (all in polynomial time), to prove the complexity of verifying a hypothesis for each framework, it suffices to show that ILP_c is DP-hard (thus also proving the hardness for each of the other frameworks) and that ILP^{s}_{LAS} is a member of DP (thus proving membership for the other frameworks). This shows that each framework is both a member of DP and also DP-hard, and therefore must be DP-complete.

Proposition 5. *Deciding verification for ILP^{s}_{LAS} is a member of DP.*

Proposition 6. *Deciding verification for ILP_c is DP-hard.*

We can now prove the complexity of deciding verification for ILP_c, ILP_{LAS} and ILP_{LOAS}. This proves the corresponding entries in Table 2.

Theorem 2. *Deciding whether a given H is a solution of any ILP_c, ILP_{LAS}, ILP_{LOAS} or $ILP^{context}_{LOAS}$ task is DP-complete in each case.*

Proof. By Proposition 6, deciding the verification for ILP_c is DP-hard. By Proposition 2, deciding the verification for ILP_c reduces to deciding verification for ILP_{LAS} which, in turn, reduces to deciding verification for $ILP^{context}_{LOAS}$, which reduces to deciding satisfiability for ILP_{LOAS}, which again reduces to deciding verification for ILP^s_{LAS} and by Proposition 5, deciding verification for ILP^s_{LAS} is a member of DP. Deciding verification for each of these learning frameworks must therefore be both a member of DP and must be DP-hard. Hence, deciding verification for each framework is DP-complete. □

Similarly, to show that deciding satisfiability is Σ^P_2-complete for each framework, we only need to show that ILP^s_{LAS} is a member of Σ^P_2 and ILP_c is Σ^P_2-hard.

Proposition 7. *Deciding satisfiability for ILP^s_{LAS} is in Σ^P_2.*

Proposition 8. *Deciding satisfiability for ILP_c is Σ^P_2-hard.*

We can now prove the complexity of deciding satisfiability for ILP_c, ILP_{LAS} and ILP_{LOAS}. This proves the remaining entries in Table 2.

Theorem 3. *Deciding the satisfiability of any ILP_c, ILP_{LAS}, ILP_{LOAS} or $ILP^{context}_{LOAS}$ task is Σ^P_2-complete in each case.*

Proof. (similar to the proof of Theorem 2) By Proposition 8, deciding satisfiability for ILP_c is Σ^P_2-hard. By Proposition 2, deciding satisfiability for ILP_c reduces to deciding satisfiability for ILP_{LAS} which, in turn, reduces to deciding satisfiability for $ILP^{context}_{LOAS}$, which reduces to deciding satisfiability for ILP_{LOAS}, which again reduces to deciding satisfiability of ILP^s_{LAS}. By Proposition 7, deciding satisfiability for ILP^s_{LAS} is in Σ^P_2. Deciding satisfiability for each of these learning frameworks is therefore both a member of Σ^P_2 and is Σ^P_2-hard. Hence, deciding satisfiability for each framework is Σ^P_2-complete. □

4.4. Considering noisy examples

Although the frameworks considered in this paper were originally presented under the assumption that all examples were perfectly labeled (i.e. there is no noise in the examples), some of the systems for solving these tasks do consider noise when searching for an optimal solution.

A common approach, used by both XHAIL [42] and ILASP [32] is to penalise hypotheses for the examples they do not cover. In ILASP, some examples can be labeled together with a penalty that must be paid if a hypothesis does not cover the example. Any example that is not labeled with a penalty must be covered by any inductive solution. Given a set of examples E, the score of a hypothesis H is said to be $|H| + p(H, E)$, where $|H|$ is the length of the hypothesis, and $p(H, E)$ is the sum of the weights of all examples that are not covered by H. As a hypothesis is an inductive solution if and only if it covers all the examples that are labeled with a penalty, that were not labeled with an explicit penalty, the two decision problems of verification and satisfiability can be reduced to the corresponding decisions for non-noisy tasks (by simply removing any example with a penalty).

5. Generality

In this section, we present a new notion of the *generality* of a learning framework. The aim is to get a sense of which class of ASP programs a framework is capable of learning, if given sufficient examples. Language biases tend, in general, to impose their own restrictions on the classes of program that can be learned. They are primarily used to aid the performance of the computation, rather than to capture intrinsic properties of a learning framework. In this section we will therefore consider learning tasks with unrestricted hypothesis spaces: hypotheses can be constructed from any set of (first order) normal rules, choice rules and hard and weak constraints. We assume each learning framework $\mathcal{F}$ to have a task consisting of a pair $\langle B, E_{\mathcal{F}} \rangle$, where B is the (first order ASP) background knowledge and $E_{\mathcal{F}}$ is a tuple consisting of the examples for this framework; for example $E_{LAS} = \langle E^+, E^- \rangle$ where E^+ and E^- are sets of partial interpretations.

Allowing an unrestricted hypothesis space raises the question of whether a learning framework is general enough to define tasks that lead to a particular set of hypotheses as the inductive solutions. On a first instance, we could say that a framework $\mathcal{F}$ is general enough to learn a hypothesis H if there is at least one task $T_{\mathcal{F}}$ in this framework such that H is an inductive solution of $T_{\mathcal{F}}$. However, as shown in Example 6, such a "loose notion" of generality may lead to the trivial learning framework, whose learning tasks have no examples, as the most general framework possible.

Example 6. Consider the trivial learning framework $ILP_\top$ whose learning tasks are pairs $\langle B, E_\top \rangle$, where $E_\top$ is the empty tuple and B is an ASP program. $ILP_\top(\langle B, E_\top \rangle)$ is then the set of all ASP programs, i.e., every ASP program is a solution of every $ILP_\top$ task. Although for every hypothesis H, given any background knowledge B there is clearly a set of examples $E_\top$ such that $H \in ILP_\top(\langle B, E_\top \rangle)$, every other possible hypothesis is also a solution of this same task, making it impossible to distinguish any hypothesis from another.

It is clearly not sufficient to say that a framework is general enough to learn some target hypothesis (denoted from now on as H_T) if we can find at least one learning task with H_T as a solution. What this definition lacks is a way to express that H_T is a solution of a task T, but that some other (unwanted) hypothesis is not a solution of T. To capture this property of a learning framework we should be able to say that a task T can *distinguish* a hypothesis H_T from the unwanted hypothesis. Pairs of target and unwanted hypotheses, which can be distinguished from each other, are an interesting starting point when considering generality of a learning framework. But this again might not be the only property of generality. Frameworks, such as brave induction, can distinguish the target hypothesis H_T from two (or more) unwanted hypotheses, e.g., H'_1 and H'_2, in two separate learning tasks, but they may not have a single learning task capable of accepting H_T as inductive solution but neither H'_1 nor H'_2. Consider for instance the following example.

Example 7. Imagine the scenario where we are observing a coin being tossed several times. Obviously there are two outcomes, and we would like to learn an ASP program whose answer sets correspond to these two different outcomes. Consider the background knowledge B to be empty, and the atoms `heads` and `tails` to be true when the coin lands on heads or tails respectively. Our target hypothesis H_T is an ASP program such that $AS(B \cup H) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$. One such hypothesis could be the program $H_T = \{1\{\texttt{heads}, \texttt{tails}\}1.\}$. Consider now the two hypotheses $H'_1 = \{\texttt{heads}.\}$ and $H'_2 = \{\texttt{tails}.\}$, which correspond to the coin always landing on heads or tails respectively. Neither of these hypothesis correctly represent the behaviour of the coin, so they are unwanted hypotheses. There is one answer set, $\{\texttt{heads}\}$, of $B \cup H'_1$ and one answer set, $\{\texttt{tails}\}$, of $B \cup H'_2$. ILP_b can distinguish H_T from H'_1 and from H'_2 separately with the tasks $\langle B, \langle \{\texttt{tails}\}, \emptyset \rangle \rangle$ and $\langle B, \langle \{\texttt{heads}\}, \emptyset \rangle \rangle$, respectively. But there is, however, no learning task for ILP_b for which H_T is an inductive solution and neither H'_1 nor H'_2 is.

A more general notion of generality of learning framework can be considered, which looks at distinguishing a target hypothesis H_T from a set of unwanted hypotheses S. In Section 5.2 we introduce the notion of *one-to-many-distinguishability class* of a learning framework. This corresponds to the class of pairs of single hypothesis H_T's and set S's of hypotheses for which a learning framework has at least one task that distinguishes H_T from each hypothesis in S. Informally, this notion expresses the generality of a framework in finding a single target hypothesis in the presence of many unwanted hypotheses. In Section 5.3, we extend one-to-many-distinguishability class of a learning framework to *many-to-many-distinguishability*, which in turns captures the notion of distinguishing a set of target hypotheses S_1 from another set of unwanted hypotheses S_2, with a single task.

In the remainder of this section we explore these three new measures of generality, expressed as three different learning problems. One-to-one-distinguishability determines the hypotheses that a framework is general enough to learn, while ruling out another unwanted hypothesis; one-to-many-distinguishability determines the hypotheses that can be learned from within a space of unwanted hypotheses; and finally, many-to-many-distinguishability determines exactly which sets of hypotheses can be learned. We will prove properties of our three classes of generalities making use of a definition of *strong reduction* from one framework to another. Strong reduction is different from the concept of reduction presented in [45]. Definitions 11 and 12 present, respectively, a reformulation of the notion of reduction introduced in [45] and of our new concept of strong reduction.

Definition 11. A framework $\mathcal{F}_1$ *reduces to* $\mathcal{F}_2$ (written $\mathcal{F}_1 \rightarrow_r \mathcal{F}_2$) if for every $\mathcal{F}_1$ task $T_{\mathcal{F}_1}$ there is an $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$ such that $ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. A framework $\mathcal{F}_1$ is *at least as r-general as* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$; and $\mathcal{F}_1$ is *more r-general than* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$ and $\mathcal{F}_1 \nrightarrow_r \mathcal{F}_2$.

Example 8. Consider the ILP_b and ILP_c learning frameworks. $ILP_b \rightarrow_r ILP_c$, as any ILP_b task $\langle B, \langle E^+, E^- \rangle \rangle$ maps to the ILP_c task $\langle B \cup \{\texttt{:- not } e. \mid e \in E^+\} \cup \{\texttt{:- } e. \mid e \in E^-\}, \langle \emptyset, \emptyset \rangle \rangle$. ILP_c does not, however, reduce to ILP_b. Consider, for instance, the ILP_c task $T_c = \langle \emptyset, \langle \{\texttt{p}\}, \emptyset \rangle \rangle$ and assume that there is a task $T_b = \langle B, \langle E^+, E^- \rangle \rangle$ such that $ILP_b(T_b) = ILP_c(T_c)$. The hypothesis $H_1 = \{\texttt{p}.\} \in ILP_c(T_c)$, and, given the assumption, H_1 is also in $ILP_b(T_b)$. But consider now the hypothesis $H_2 = \{0\{\texttt{p}\}1.\}$. Since $AS(B \cup H_1) \subseteq AS(B \cup H_2)$, if $B \cup H_1$ has an answer set extending $\langle E^+, E^- \rangle$, then so does $B \cup H_2$. Thus, if $H_1 \in ILP_b(T_b)$ then $H_2 \in ILP_b(T_b)$. But, although $H_2 \in ILP_b(T_b)$, it is easy to see that $H_2 \notin ILP_c(T_c)$, so making $ILP_b(T_b)$ not equal to $ILP_c(T_c)$. Hence, ILP_c does not reduce to ILP_b, and ILP_c is more r-general than ILP_b.

We discuss the relationship between reductions and our own measures of generality in Section 6. Our notion of strong reduction differs from the above notion of reduction, in the fact that the reduced task must have the same background knowledge as the original task.

Definition 12. A framework $\mathcal{F}_1$ *strongly reduces to* $\mathcal{F}_2$ (written $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$) if for every $\mathcal{F}_1$ task $T_{\mathcal{F}_1} = \langle B, E_{\mathcal{F}_1} \rangle$ there is an $\mathcal{F}_2 = \langle B, E_{\mathcal{F}_2} \rangle$ task $T_{\mathcal{F}_2}$ such that $ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. A framework $\mathcal{F}_1$ is *at least as sr-general as* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$; and $\mathcal{F}_1$ is *more sr-general than* $\mathcal{F}_2$ if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ and $\mathcal{F}_1 \nrightarrow_{sr} \mathcal{F}_2$.

Proposition 9 shows the strong reduction relations between the frameworks considered in this paper. Note that although ILP_c is more *r*-general than ILP_b (as shown in Example 8), it is not more *sr*-general than ILP_b. This is because without changing the background knowledge, ILP_c cannot represent the same ILP_b tasks.

Proposition 9.

1. $ILP_b \rightarrow_{sr} ILP_{sm} \rightarrow_{sr} ILP_{LAS} \rightarrow_{sr} ILP_{LOAS} \rightarrow_{sr} ILP^{context}_{LOAS}$
2. $ILP_c \rightarrow_{sr} ILP_{LAS}$

Proof.

1. For any ILP_b task $T_b = \langle B, \langle E^+, E^- \rangle \rangle$, $ILP_b(T_b) = ILP_{sm}(\langle B, \langle \{\langle E^+, E^- \rangle\} \rangle \rangle)$
   For any ILP_{sm} task $T_{sm} = \langle B, \langle \{e_1, \ldots, e_n\} \rangle \rangle$, $ILP_{sm}(T_{sm}) = ILP_{LAS}(\langle B, \langle \{e_1, \ldots, e_n\}, \emptyset \rangle \rangle)$
   For any ILP_{LAS} task $T_{LAS} = \langle B, \langle E^+, E^- \rangle \rangle$, $ILP_{LAS}(T_{LAS}) = ILP_{LOAS}(\langle B, \langle E^+, E^-, \emptyset, \emptyset \rangle \rangle)$
   For any partial interpretation e, let $c(e)$ be the CDPI $\langle e, \emptyset \rangle$. For any ILP_{LOAS} task $T_{LOAS} = \langle B, \langle E^+, E^-, O^b, O^c \rangle \rangle$, $ILP_{LOAS}(T_{LOAS}) = ILP^{context}_{LOAS}(\langle B, \langle \{c(e) \mid e \in E^+\}, \{c(e) \mid e \in E^-\}, \{\langle c(e_1), c(e_2) \rangle \mid \langle e_1, e_2 \rangle \in O^b\}, \{\langle c(e_1), c(e_2) \rangle \mid \langle e_1, e_2 \rangle \in O^c\} \rangle \rangle)$
2. For any ILP_c task $T_c = \langle B, \langle \{e^+_1, \ldots, e^+_m\}, \{e^-_1, \ldots, e^-_n\} \rangle \rangle$, $ILP_c(T_c) = ILP_{LAS}(\langle B, \langle \{\langle \emptyset, \emptyset \rangle\}, \{\langle \emptyset, \{e^+_1\} \rangle, \ldots, \langle \emptyset, \{e^+_m\} \rangle, \langle \{e^-_1\}, \emptyset \rangle, \ldots, \langle \{e^-_n\}, \emptyset \rangle \} \rangle \rangle)$. Note that the empty ILP_{LAS} positive example enforces that there is at least one answer set, and both the ILP_c positive and negative examples are mapped to ILP_{LAS} negative examples which enforce in the case of positive (resp. negative) examples that they are not false (resp. not true) in any answer set, and hence true (resp. false) in every answer set. □

5.1. Distinguishability

A one-to-one-distinguishability class captures those pairs of hypotheses H_1 and H_2 that can be distinguished from each other with respect to a given possible background knowledge.

Definition 13. The *one-to-one-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^1_1(\mathcal{F})$) is the set of tuples $\langle B, H_1, H_2 \rangle$ of ASP programs for which there is at least one task $T_{\mathcal{F}} = \langle B, E_{\mathcal{F}} \rangle$ such that $H_1 \in \mathcal{F}(T_{\mathcal{F}})$ and $H_2 \notin \mathcal{F}(T_{\mathcal{F}})$. For each $\langle B, H_1, H_2 \rangle \in \mathcal{D}^1_1(\mathcal{F})$, $T_{\mathcal{F}}$ is said to *distinguish* H_1 from H_2 with respect to B. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^1_1$-general as (resp. than) $\mathcal{F}_2$ if $\mathcal{D}^1_1(\mathcal{F}_2) \subseteq \mathcal{D}^1_1(\mathcal{F}_1)$ (resp. $\mathcal{D}^1_1(F_2) \subset \mathcal{D}^1_1(F_1)$).

Note that the one-to-one-distinguishability relationship is not symmetric; i.e. there are pairs of hypotheses H_1 and H_2 such that, given a background knowledge B, H_1 can be distinguished from H_2, but H_2 can not be distinguished from H_1. This is illustrated by Example 9.

Example 9. Consider a background knowledge B that defines the concepts of cell, same_block, same_row and same_column for a `4x4` Sudoku grid.

Let H_1 be the incomplete description of the Sudoku rules:

```
1 { value(C, 1), value(C, 2), value(C, 3), value(C, 4) } 1 :- cell(C).
:- value(C1, V), value(C2, V), same_row(C1, C2).
:- value(C1, V), value(C2, V), same_col(C1, C2).
```

Also let H_2 be the complete description of the Sudoku rules:

```
1 { value(C, 1), value(C, 2), value(C, 3), value(C, 4) } 1 :- cell(C).
:- value(C1, V), value(C2, V), same_row(C1, C2).
:- value(C1, V), value(C2, V), same_col(C1, C2).
:- value(C1, V), value(C2, V), same_block(C1, C2).
```

ILP_b can distinguish H_1 from H_2 with respect to B. This can be seen using the task $\langle B, \langle \{$`value((1,1),1)`, `value((2,2),1)`$\}, \emptyset \rangle \rangle$. On the other hand, ILP_b cannot distinguish H_2 from H_1. Whatever examples are given in a learning task to learn H_2, it must be the case that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$, where A is an answer set of $B \cup H_2$. But answer sets of $B \cup H_2$ are also answer sets of $B \cup H_1$. So A is also an answer set of $B \cup H_1$, which implies that H_1 satisfies the same examples and is a solution of the same learning task.

Table 3
A summary of the sufficient and necessary conditions in each learning framework for a hypothesis H_1 to be distinguishable from another hypothesis H_2 with respect to a background knowledge B.

Framework $\mathcal{F}$	Sufficient/necessary condition for $\langle B, H_1, H_2\rangle$ to be in $\mathcal{D}^1_1(\mathcal{F})$
$ILP_\top$	$\bot$
ILP_b	$AS(B \cup H_1) \not\subseteq AS(B \cup H_2)$
ILP_{sm}	$AS(B \cup H_1) \not\subseteq AS(B \cup H_2)$
ILP_c	$AS(B \cup H_1) \neq \emptyset \wedge (AS(B \cup H_2) = \emptyset \vee (\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)))$
ILP_{LAS}	$AS(B \cup H_1) \neq AS(B \cup H_2)$
ILP_{LOAS}	$(AS(B \cup H_1) \neq AS(B \cup H_2)) \vee (ord(B \cup H_1) \neq ord(B \cup H_2))$
$ILP^{context}_{LOAS}$	$(B \cup H_1 \not\equiv^s B \cup H_2) \vee (\exists C \in \mathcal{ASP}^{ch}$ st $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C))$

In fact, Proposition 10 generalises Example 9 showing that ILP_b cannot distinguish any program containing a constraint from the same program without the constraint.

Proposition 10. *ILP_b cannot distinguish any hypothesis H which contains a constraint C from $H \backslash \{C\}$, with respect to any background knowledge.*

Proof. Assume for contradiction that there is a hypothesis $H = H' \cup C$ where C is a constraint and an ILP_b task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $H' \notin ILP_b(T_b)$.

$\Rightarrow$ $\exists A \in AS(B \cup H)$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$. But as C is a constraint $AS(B \cup H) \subseteq AS(B \cup H')$ and so $A \in AS(B \cup H')$.
$\Rightarrow$ $\exists A \in AS(B \cup H')$ such that $E^+ \subseteq A$ and $E^- \cap A = \emptyset$.
$\Rightarrow$ $H' \in ILP_b(T_b)$. Contradiction! $\square$

One useful property is that if there is a strong reduction from one framework $\mathcal{F}_1$ to another framework $\mathcal{F}_2$ then $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. Note that $\mathcal{F}_2$ is not guaranteed to be more $\mathcal{D}^1_1$-general than $\mathcal{F}_1$, even in the case when there is no reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$.

Proposition 11. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$.*

Proof. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Take any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$. There must be some task $T_{\mathcal{F}_1}$, with background knowledge B, such that $H_1 \in ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $H_2 \notin ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there must be some task $T_{\mathcal{F}_2}$, with background knowledge B, such that $H_1 \in ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $H_2 \notin ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$. Hence, $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. $\square$

As there are clear strong reductions (shown by Proposition 9), an ordering of the one-to-one-distinguishability classes of the frameworks emerges (shown in Corollary 4).

Corollary 4.

1. $\mathcal{D}^1_1(ILP_b) \subseteq \mathcal{D}^1_1(ILP_{sm}) \subseteq \mathcal{D}^1_1(ILP_{LAS}) \subseteq \mathcal{D}^1_1(ILP_{LOAS}) \subseteq \mathcal{D}^1_1(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_1(ILP_c) \subseteq \mathcal{D}^1_1(ILP_{LAS})$

While this does give us information about the ordering of the power of the frameworks to distinguish between hypotheses, it does not tell us, for example, what the relationship is between the distinguishability classes of ILP_b and ILP_c. It does not tell us which of the $\subseteq$'s are strict (in fact, $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$, but the rest are strict subset relations). For each framework, Table 3 shows the necessary and sufficient condition needed to be able to distinguish hypotheses. In the case of the cautious induction framework, the condition makes use of a new notation. Given a program P, $\mathcal{E}_b(P) = \{\mathtt{i}_1 \wedge \ldots \wedge \mathtt{i}_m \wedge \mathtt{not}\ \mathtt{e}_1 \wedge \ldots \wedge \mathtt{not}\ \mathtt{e}_n | \exists A \in AS(P)$ st $\mathtt{i}_1, \ldots, \mathtt{i}_m \in A$ and $\mathtt{e}_1, \ldots, \mathtt{e}_n \notin A\}$, i.e. $\mathcal{E}_b(P)$ denotes the set of conjunctions of literals that are true in at least one answer set of P. Similarly, we use $\mathcal{E}_c(P)$ to denote the set of conjunctions of literals that are true in every answer set of P. The following property holds.

Proposition 12. *For any programs P_1 and P_2, $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$ if and only if $AS(P_1) \subseteq AS(P_2)$.*

Propositions 13 to 18 prove the one-to-one-distinguishability classes of ILP_b, ILP_{sm}, ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$, showing also the sufficient and necessary conditions for distinguishability presented in Table 3. To aid readability, the proofs are in the appendix rather than the main paper.

Proposition 13. $\mathcal{D}^1_1(ILP_b) = \left\{\langle B, H_1, H_2\rangle \middle| AS(B \cup H_1) \not\subseteq AS(B \cup H_2)\right\}$.

Interestingly, although $ILP_{sm} \nrightarrow_{sr} ILP_b$, $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$. This is shown by Proposition 14. The reason for this is that if ILP_{sm} can distinguish one hypothesis H_1 from another hypothesis H_2 then, there must be some task T_{sm} such that H_1 is a solution of T_{sm} and H_2 is not. This means that H_1 must cover all of the examples of T_{sm} and there must be at least one (partial interpretation) example of T_{sm} which is not covered by H_2. This partial interpretation example can be given as the set of positive and negative examples in an ILP_b task. This ILP_b task will then distinguish H_1 from H_2.

Proposition 14. $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$.

To better compare the conditions for ILP_b and ILP_c, we can express the necessary and sufficient condition of ILP_b in terms of the notion $\mathcal{E}_b(P)$. Specifically, in ILP_b for one hypothesis H_1 to be distinguishable from another hypothesis H_2 (with respect to a background knowledge B) it is both necessary and sufficient for $\mathcal{E}_b(B \cup H_1)$ to contain at least one conjunction that is not in $\mathcal{E}_b(B \cup H_2)$. This is because the extra conjunction can be used to generate a set of examples that are covered by H_1 but not H_2. This is demonstrated by Example 10.

Example 10. Consider again the programs $B = \emptyset$, $H_1 = \{1\{\texttt{heads}, \texttt{tails}\}1\,.\}$ and $H_2 = \{\texttt{heads}.\}$. $\mathcal{E}_b(B \cup H_1)$ contains the conjunction $\texttt{not heads} \wedge \texttt{tails}$, whereas $\mathcal{E}_b(B \cup H_2)$ does not. This conjunction can be mapped into the positive example `tails` and the negative example `heads`, which $B \cup H_1$ covers, but $B \cup H_2$ does not – i.e. the task $\langle B, \langle\{\texttt{tails}\}, \{\texttt{heads}\}\rangle\rangle$ distinguishes H_1 from H_2.

So, as the one-to-one-distinguishability condition for ILP_b could also be expressed as $\mathcal{E}_b(B \cup H_1) \not\subseteq \mathcal{E}_b(B \cup H_2)$, it might be expected that the one-to-one-distinguishability condition for ILP_c would be that $\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$. Indeed this would be the case, if it were not for the extra condition that ILP_c imposes on any inductive solution: that is, any inductive solution H must be such that $B \cup H$ is satisfiable. Although this extra condition may seem unnecessary at first sight, its importance becomes clear when considering distinguishability. Without this extra condition, no hypothesis would be distinguishable from the hypothesis given by the empty constraint "`:- .`" – i.e. there would be no hypothesis H such that $\langle B, H, \{\texttt{:- .}\}\rangle \in \mathcal{D}^1_1(ILP_c)$ (for any B). This is because there cannot be any answer set of $B \cup \{\texttt{:- .}\}$ that does not cover the examples (as there are no answer sets). As ILP_c has the extra condition that $B \cup H$ must be satisfiable, its distinguishability condition is slightly more complicated than $\mathcal{E}_c(B \cup H_1) \not\subseteq \mathcal{E}_c(B \cup H_2)$, as shown in Proposition 15.

Proposition 15. $\mathcal{D}^1_1(ILP_c) = \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{c} AS(B \cup H_1) \neq \emptyset \wedge \\ (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_2) \not\subseteq \mathcal{E}_c(B \cup H_1)) \end{array} \right\}$.

We now prove the one-to-one-distinguishability classes of our own frameworks, ILP_{LAS} and ILP_{LOAS}. $\mathcal{D}^1_1(ILP_{LAS})$ contains both $\mathcal{D}^1_1(ILP_b)$ and $\mathcal{D}^1_1(ILP_c)$ as ILP_{LAS} can distinguish any two hypotheses which, combined with the background knowledge, have different answer sets.

Proposition 16. $\mathcal{D}^1_1(ILP_{LAS}) = \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$.

As shown in Theorem 4, ILP_{LOAS} is more $\mathcal{D}^1_1$-general than ILP_{LAS}. This is because ILP_{LOAS} is able to use its ordering examples to distinguish any two hypotheses that, when combined with the background knowledge, order their answer sets differently, even if the two programs have the same answer sets.

Proposition 17. $\mathcal{D}^1_1(ILP_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \;\middle|\; \begin{array}{l} AS(B \cup H_1) \neq AS(B \cup H_2) \; or \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$.

Note that we assume ILP_{LOAS} to be able to give ordering examples with any of the binary ordering operators. The slightly more restrictive version of ILP_{LOAS}, presented in [23] where the operator is only the $<$, has a smaller one-to-one-distinguishability class. This is shown in Example 11.

Example 11. Consider the heads and tails problem again, where $B = \{\ 1\{heads, tails\}1\ .\ \}$, and two potential hypotheses:

- $H_1 = \emptyset$
- $H_2 = \{:\sim\ heads\ .\ [1@1]\}$

$AS(B \cup H_1) = AS(B \cup H_2) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$. If we consider the restricted ILP_{LOAS} where only the operator $<$ is used to express ordering over the examples, H_2 can be distinguished from H_1, but not H_1 from H_2. This is because all answer sets of $B \cup H_1$ are equally optimal – neither $\langle\{\texttt{tails}\}, \{\texttt{heads}\}, <\rangle$ nor $\langle\{\texttt{heads}\}, \{\texttt{tails}\}, <\rangle$ is in $ord(B \cup H_1)$. In contrast, if we allow the use of any of the binary ordering operators, we can consider a task with the ordering example $\langle\{\texttt{tails}\}, \{\texttt{heads}\}, =\rangle$ and be able to distinguish H_1 from H_2. The learned hypothesis H_1 has no weak constraints, so

the two answer sets are equally optimal and the ordering example is respected by H_1, whereas H_2 prefers {tails} to {heads}.

ILP_{LOAS} can distinguish any two hypotheses that, when combined with a fixed background knowledge, behave differently. It cannot distinguish hypotheses that are different but behave the same with respect to the background knowledge. This means that there are some hypotheses that are not strongly equivalent (when combined with the background knowledge), but ILP_{LOAS} cannot distinguish one from the other. We now show that $ILP^{context}_{LOAS}$ *can* distinguish between any two hypotheses, H_1 and H_2, that, when combined with the background knowledge, are not strongly equivalent, or there is at least one program $C \in \mathcal{ASP}^{ch}$ (consisting of normal rules, choice rules and hard constraints), such that $ord(B \cup H_1 \cup C) \neq ord(B, \cup H_2 \cup C)$.

Proposition 18.

$$\mathcal{D}^1_1(ILP^{context}_{LOAS}) = \left\{ \langle B, H_1, H_2 \rangle \;\middle|\; \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \text{ such that } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$$

Now that we have proven the distinguishability classes for each learning framework, we can strengthen the statement of Corollary 4 and more precisely state the relationship between the distinguishability classes of the frameworks. Apart from the case of ILP_b and ILP_{sm}, each of the subset relations in Corollary 4 are in fact strict subsets.

Theorem 4. *Consider the learning frameworks ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.*

1. $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm}) \subset \mathcal{D}^1_1(ILP_{LAS}) \subset \mathcal{D}^1_1(ILP_{LOAS}) \subset \mathcal{D}^1_1(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_1(ILP_c) \subset \mathcal{D}^1_1(ILP_{LAS})$

Proof.

1. The fact that $\mathcal{D}^1_1(ILP_b) = \mathcal{D}^1_1(ILP_{sm})$ was shown in Proposition 14. By Corollary 4, $\mathcal{D}^1_1(ILP_{sm}) \subseteq \mathcal{D}^1_1(ILP_{LAS}) \subseteq \mathcal{D}^1_1(ILP_{LOAS}) \subseteq \mathcal{D}^1_1(ILP^{context}_{LOAS})$; hence, it remains to show that $\mathcal{D}^1_1(ILP_{sm}) \neq \mathcal{D}^1_1(ILP_{LAS}) \neq \mathcal{D}^1_1(ILP_{LOAS}) \neq \mathcal{D}^1_1(ILP^{context}_{LOAS})$
   - Consider the tuple $\langle B, H_1, H_2 \rangle$, where $B = \emptyset$, $H_1 = \{\texttt{p.}\}$ and $H_2 = \{\texttt{1\{p,q\}1}\}$. $AS(B \cup H_1) \subset AS(B \cup H_2)$, hence $\langle B, H_1, H_2 \rangle$ does not satisfy the condition, given in Table 3, necessary for it to be in $\mathcal{D}^1_1(ILP_{sm})$. It does, however, satisfy the condition for it to be in $\mathcal{D}^1_1(ILP_{LAS})$. Hence, $\mathcal{D}^1_1(ILP_{sm}) \neq \mathcal{D}^1_1(ILP_{LAS})$.
   - Consider the tuple $\langle B, H_1, H_2 \rangle$, where $B = \{\texttt{1\{p,q\}1}\}$, $H_1 = \emptyset$ and $H_2 = \{\texttt{:\sim\ p.[1@1]}\}$. $AS(B \cup H_1) = AS(B \cup H_2)$ and $ord(B \cup H_1) \neq ord(B \cup H_2)$. Hence, by the conditions in Table 3, $\langle B, H_1, H_2 \rangle$ is in $\mathcal{D}^1_1(ILP_{LOAS})$ but is not in $\mathcal{D}^1_1(ILP_{LAS})$. Therefore, $\mathcal{D}^1_1(ILP_{LAS}) \neq \mathcal{D}^1_1(ILP_{LOAS})$.
   - Consider the tuple $\langle B, H_1, H_2 \rangle$, where $B = \emptyset$, $H_1 = \emptyset$ and $H_2 = \{\texttt{p:-q.}\}$. Also consider the program $P = \{\texttt{q.}\}$. $AS(B \cup H_1) = AS(B \cup H_2)$ and $ord(B \cup H_1) = ord(B \cup H_2)$, but $AS(B \cup H_1 \cup P) \neq AS(B \cup H_2 \cup P)$; this shows that $B \cup H_1 \not\equiv_s B \cup H_2$. Hence, by the conditions in Table 3, $\langle B, H_1, H_2 \rangle$ is in $\mathcal{D}^1_1(ILP^{context}_{LOAS})$ but is not in $\mathcal{D}^1_1(ILP_{LOAS})$. Therefore, $\mathcal{D}^1_1(ILP_{LOAS}) \neq \mathcal{D}^1_1(ILP^{context}_{LOAS})$.
2. By Corollary 4, $\mathcal{D}^1_1(ILP_c) \subseteq \mathcal{D}^1_1(ILP_{LAS})$. Hence, it remains to show that $\mathcal{D}^1_1(ILP_c) \neq \mathcal{D}^1_1(ILP_{LAS})$. Consider the tuple $\langle B, H_1, H_2 \rangle$, where $B = \{\texttt{p:- not p}\}$, $H_1 = \emptyset$ and $H_2 = \{\texttt{p.}\}$. $AS(B \cup H_1) = \emptyset$ and $AS(B \cup H_1) \neq AS(B \cup H_2)$. By the conditions in Table 3, $\langle B, H_1, H_2 \rangle$ is in $\mathcal{D}^1_1(ILP_{LAS})$ but is not in $\mathcal{D}^1_1(ILP_c)$. Hence, $\mathcal{D}^1_1(ILP_c) \neq \mathcal{D}^1_1(ILP_{LAS})$. □

5.2. The one-to-many-distinguishability class of a learning framework

In practice an ILP task has a search space of possible hypotheses, and it is important to know the cases in which one particular hypothesis can be distinguished from the rest. In what follows, we analyse the conditions under which a learning framework can distinguish an hypothesis from *a set* of other hypotheses. As mentioned at the beginning of Section 5, this corresponds to the new notion we call the *one-to-many-distinguishability class* of a learning framework, which is a generalisation of the notion of the *one-to-one-distinguishability class* described above.

Definition 14. The *one-to-many-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^1_m(\mathcal{F})$) is the set of all tuples $\langle B, H, \{H_1, \ldots, H_n\} \rangle$ such that there is a task $T_{\mathcal{F}}$ which distinguishes H from each H_i with respect to B. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ if $\mathcal{D}^1_m(\mathcal{F}_2) \subseteq \mathcal{D}^1_m(\mathcal{F}_1)$ (resp. $\mathcal{D}^1_m(F_2) \subset \mathcal{D}^1_m(F_1)$).

The one-to-many-distinguishability class tells us the circumstances in which a framework is general enough to distinguish some target hypothesis from a set of unwanted hypotheses. Note that, although the tuples in a one-to-

many-distinguishability class that have a singleton set as third argument correspond to the tuples in a one-to-one-distinguishability class of that framework, it is not always the case that if $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ then $\mathcal{F}_1$ is also more $\mathcal{D}^1_1$-general than $\mathcal{F}_2$. For example, we will see that ILP_{sm} is more $\mathcal{D}^1_m$-general than ILP_b, but we have already shown in Proposition 14 that the ILP_b and ILP_{sm} are equally $\mathcal{D}^1_1$-general. Proposition 19 shows, however, that if $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$ then $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$.

Proposition 19. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ such that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$, $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ (i.e. $\mathcal{D}^1_m(\mathcal{F}_2) \subseteq \mathcal{D}^1_m(\mathcal{F}_1) \Rightarrow \mathcal{D}^1_1(\mathcal{F}_2) \subseteq \mathcal{D}^1_1(\mathcal{F}_1)$).*

Proof. Assume that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$ and let $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$. To show that $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$, we must show that $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$.

As $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_2)$, $\langle B, H_1, \{H_2\}\rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$; hence, as $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$, $\langle B, H_1, \{H_2\}\rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$; and hence, $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(\mathcal{F}_1)$. □

We have already seen that if there is a strong reduction from $\mathcal{F}_1$ to $\mathcal{F}_2$ then $\mathcal{F}_2$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_1$. Proposition 20 shows that a similar result holds for $\mathcal{D}^1_m$-generality. Similarly to $\mathcal{D}^1_1$-generality, however, a strong reduction from $\mathcal{F}_1$ to $\mathcal{F}_2$ does not imply that $\mathcal{F}_2$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_1$, even in the case that there is no strong reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$.

Proposition 20. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$.*

Proof. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Take any $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$. There must be some task $T_{\mathcal{F}_1}$, with background knowledge B, such that $H \in ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $S \cap ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = \emptyset$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there must be some $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$, with background knowledge B, such that $H \in ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $S \cap ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = \emptyset$. So $\langle B, H, S\rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$. Hence, $\mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$. □

Due to the strong reductions shown in Proposition 9, an ordering of the one-to-many-distinguishability classes of the frameworks emerges (shown in Corollary 5).

Corollary 5.

1. $\mathcal{D}^1_m(ILP_b) \subseteq \mathcal{D}^1_m(ILP_{sm}) \subseteq \mathcal{D}^1_m(ILP_{LAS}) \subseteq \mathcal{D}^1_m(ILP_{LOAS}) \subseteq \mathcal{D}^1_m(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_m(ILP_c) \subseteq \mathcal{D}^1_m(ILP_{LAS})$

This time, we will see that each of the $\subseteq$'s in Corollary 5 can be upgraded to a strict $\subset$. Rather than proving the one-to-many-distinguishability classes from scratch, we now present a useful result. For some frameworks, the one-to-one-distinguishability class of a learning framework can be used to construct the one-to-many-distinguishability class. This is the case when the framework has *closed one-to-many-distinguishability* (formalised by Definition 15). Proposition 21 and Corollary 6 show how the one-to-many-distinguishability class of a framework can be constructed using its one-to-one-distinguishability class if it has closed one-to-many-distinguishability.

Definition 15. Given any learning framework $\mathcal{F}$, the *closure* of the one-to-many-distinguishability class, written $\overline{\mathcal{D}^1_m(\mathcal{F})}$, is the set $\{\langle B, H, S_1 \cup \ldots \cup S_n\rangle \mid \langle B, H, S_1\rangle, \ldots, \langle B, H, S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})\}$. We say that $\mathcal{F}$ has *closed one-to-many-distinguishability* if and only if $\overline{\mathcal{D}^1_m(\mathcal{F})} = \mathcal{D}^1_m(\mathcal{F})$.

Proposition 21. *For any learning framework $\mathcal{F}$, $\overline{\mathcal{D}^1_m(\mathcal{F})} = \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$.*

Corollary 6. *For any learning framework $\mathcal{F}$, $\mathcal{D}^1_m(\mathcal{F}) \subseteq \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$. The equality holds if and only if $\mathcal{F}$ has closed one-to-many-distinguishability.*

Note that not all learning frameworks have closed one-to-many-distinguishability; for instance, Example 12 shows that brave induction does not. We will show that induction of stable models, on the other hand, does have closed one-to-many-distinguishability.

Example 12. ILP_b does not have closed one-to-many-distinguishability. We can see this by reconsidering the programs $B = \emptyset$, $H = \{$`1{heads, tails}1.`$\}$, $H_1 = \{$`heads.`$\}$ and $H_2 = \{$`tails.`$\}$. $\langle B, H, \{H_1\}\rangle \in \mathcal{D}^1_m(ILP_b)$ ($\langle B, \langle\{$`tails`$\}, \emptyset\rangle\rangle$ distinguishes H from H_1 wrt the background knowledge B). Similarly $\langle B, H, \{H_2\}\rangle \in \mathcal{D}^1_m(ILP_b)$ ($\langle B, \langle\{$`heads`$\}, \emptyset\rangle\rangle$ distinguishes H from H_2 wrt the background knowledge B). If ILP_b had closed one-to-many-distinguishability then $\langle B, H, \{H_1, H_2\}\rangle$ would

be in $\mathcal{D}^1_m(ILP_b)$; hence, to show that ILP_b does not have closed one-to-many-distinguishability it is sufficient to show that $\langle B, H, \{H_1, H_2\}\rangle \notin \mathcal{D}^1_m(ILP_b)$. Hence it remains to show that there is no task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$.

Assume for contradiction that there is such a task T_b. As $H \in ILP_b(T_b)$ and $AS(B \cup H) = \{\{\texttt{heads}\}, \{\texttt{tails}\}\}$, $E^+ \subset \{\texttt{heads}, \texttt{tails}\}$ and $E^- \subset \{\texttt{heads}, \texttt{tails}\}$ (neither can be equal to $\{\texttt{heads}, \texttt{tails}\}$ or H would not be a solution).

Case 1: $E^+ = \emptyset$
- **Case a:** $E^- = \emptyset$
  Then H_1 and H_2 would be inductive solutions. This is a contradiction as $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$.
- **Case b:** $E^- = \{\texttt{heads}\}$
  Then H_2 would be an inductive solution of T_b. Contradiction.
- **Case c:** $E^- = \{\texttt{tails}\}$
  Then H_1 would be an inductive solution of T_b. Contradiction.

Case 2: $E^+ = \{\texttt{heads}\}$
$\texttt{heads} \notin E^-$ as otherwise the task would have no solutions (and we know that H is a solution). In this case H_1 would be an inductive solution (regardless of what else is in E^-). Contradiction.

Case 3: $E^+ = \{\texttt{tails}\}$
Similarly to above case, $\texttt{tails} \notin E^-$ as otherwise the task would have no solutions. In this case H_2 would be an inductive solution (regardless of what else is in E^-). Contradiction.

Hence, there is no such task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_b(T_b)$ and $\{H_1, H_2\} \cap ILP_b(T_b) = \emptyset$. ILP_b does not have closed one-to-many-distinguishability.

In contrast to ILP_b, ILP_{sm} (which we will see does have closed one-to-many-distinguishability), can distinguish H from H_1 and H_2 with the task $\langle B, \langle\{\langle\{\texttt{heads}\}, \emptyset\rangle, \langle\{\texttt{tails}\}, \emptyset\rangle\}\rangle\rangle$. Note that this is a combination of the two brave tasks which distinguish H from H_1 and from H_2. We will show that the ability to combine tasks in this way is a sufficient condition for a framework to have closed one-to-many-distinguishability. Proposition 22 shows the one-to-many-distinguishability class of ILP_b.

Proposition 22. $\mathcal{D}^1_m(ILP_b) = \left\{\langle B, H, \{h_1, \ldots, h_m\}\rangle \middle| AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)\right\}$.

Proof.

1. Let $B, H, h_1, \ldots, h_m$ be ASP programs such that $AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)$. This implies that there is an interpretation A that is an answer set of $B \cup H$ but not an answer set of any of the programs $B \cup h_1, \ldots, B \cup h_m$. Let L be the set of atoms which occur in at least one answer set of at least one of the programs $B \cup H, B \cup h_1, \ldots B \cup h_m$; then $B \cup H$ has an answer set that extends $\langle A, L \backslash A\rangle$, but none of $B \cup h_1, \ldots B \cup h_m$ do. So the task $\langle B, \langle A, L \backslash A\rangle\rangle$ distinguishes H from h_1 to h_m. Hence, $\langle B, H, \{h_1, \ldots, h_m\}\rangle \in \mathcal{D}^1_m(ILP_b)$.
2. Assume $\langle B, H, \{h_1, \ldots, h_m\}\rangle \in \mathcal{D}^1_m(ILP_b)$. Then there is an ILP_b task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $B \cup H$ has an answer set extending $\langle E^+, E^-\rangle$ and none of $B \cup h_1, \ldots, B \cup h_m$ do. Hence, there must be at least one answer set of $B \cup H$, which is not an answer set of any of $B \cup h_1, \ldots, B \cup h_m$. Therefore $AS(B \cup H) \not\subseteq AS(B \cup h_1) \cup \ldots \cup AS(B \cup h_m)$. □

For a framework $\mathcal{F}$ to have closed one-to-many-distinguishability it is sufficient (but not necessary) that for every two $\mathcal{F}$ tasks, there is a third $\mathcal{F}$ task whose solutions are exactly those hypotheses which are solutions to both of the original two tasks. This is formalised and proved in Lemma 2. This condition is not necessary in general, but it holds for the frameworks considered in this paper that have closed one-to-many-distinguishability.

Lemma 2. *For any learning framework $\mathcal{F}$ to have closed one-to-many-distinguishability, it is sufficient that for every pair of learning tasks $T^1_{\mathcal{F}} = \langle B, E^1_{\mathcal{F}}\rangle$ and $T^2_{\mathcal{F}} = \langle B, E^2_{\mathcal{F}}\rangle$ it is possible to construct a new learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}}\rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$.*

Proof. Assume that for every pair of learning tasks $T^1_{\mathcal{F}} = \langle B, E^1_{\mathcal{F}}\rangle$ and $T^2_{\mathcal{F}} = \langle B, E^2_{\mathcal{F}}\rangle$ it is possible to construct a new learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}}\rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$. Let $\langle B, H, S_1\rangle, \ldots, \langle B, H, S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})$. To prove that $\mathcal{F}$ has closed one-to-many-distinguishability, we must show that $\langle B, H, S_1 \cup \ldots \cup S_n\rangle \in \mathcal{D}^1_m(\mathcal{F})$. We prove this by showing (by mathematical induction) that for each $k \in [1..n]$, $\langle B, H, S_1 \cup \ldots \cup S_k\rangle \in \mathcal{D}^1_m(\mathcal{F})$.

Base Case: $k = 1$. $\langle B, H, S_1\rangle \in \mathcal{D}^1_m(\mathcal{F})$ by the initial assumptions.
Inductive Hypothesis: Assume that for some $0 \leq k < n$, $\langle B, H, S_1 \cup \ldots \cup S_k\rangle \in \mathcal{D}^1_m(\mathcal{F})$.
Inductive Step: We must show that $\langle B, H, S_1 \cup \ldots \cup S_{k+1}\rangle \in \mathcal{D}^1_m(\mathcal{F})$.

As $\langle B, H, S_1 \cup \ldots \cup S_k \rangle \in \mathcal{D}^1_m(\mathcal{F})$ (by the inductive hypothesis), there must be a learning task $T^1_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T^1_{\mathcal{F}})$ and $(S_1 \cup \ldots \cup S_k) \cap ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) = \emptyset$. As $\langle B, H, S_{k+1} \rangle \in \mathcal{D}^1_m(\mathcal{F})$, there must also be a learning task $T^2_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$ and $S_{k+1} \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}}) = \emptyset$. By our initial assumption, there is a learning task $T^3_{\mathcal{F}} = \langle B, E^3_{\mathcal{F}} \rangle$ such that $ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = ILP_{\mathcal{F}}(T^1_{\mathcal{F}}) \cap ILP_{\mathcal{F}}(T^2_{\mathcal{F}})$. So, $H \in ILP_{\mathcal{F}}(T^3_{\mathcal{F}})$, $(S_1 \cup \ldots \cup S_k) \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$ and $S_{k+1} \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$. Therefore, $H \in ILP_{\mathcal{F}}(T^3_{\mathcal{F}})$ and $(S_1 \cup \ldots \cup S_{k+1}) \cap ILP_{\mathcal{F}}(T^3_{\mathcal{F}}) = \emptyset$. Hence, $\langle B, H, S_1 \cup \ldots \cup S_{k+1} \rangle \in \mathcal{D}^1_m(\mathcal{F})$. □

Proposition 23. *ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ all have closed one-to-many-distinguishability.*

Theorem 5. *Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, $\overline{\mathcal{D}^1_m(\mathcal{F}_1)} \subseteq \overline{\mathcal{D}^1_m(\mathcal{F}_2)}$ if and only if $\mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$.*

Proof. $\overline{\mathcal{D}^1_m(\mathcal{F}_1)} \subseteq \overline{\mathcal{D}^1_m(\mathcal{F}_2)}$

$$\Leftrightarrow \left\{ \langle B, H, \{H_1, \ldots, H_n\} \rangle \;\middle|\; \begin{array}{c} \langle B, H, H_1 \rangle \in \mathcal{D}^1_1(\mathcal{F}_1) \\ \ldots \\ \langle B, H, H_n \rangle \in \mathcal{D}^1_1(\mathcal{F}_1) \end{array} \right\}$$

$$\subseteq \left\{ \langle B, H, \{H_1, \ldots, H_n\} \rangle \;\middle|\; \begin{array}{c} \langle B, H, H_1 \rangle \in \mathcal{D}^1_1(\mathcal{F}_2) \\ \ldots \\ \langle B, H, H_n \rangle \in \mathcal{D}^1_1(\mathcal{F}_2) \end{array} \right\} \text{(by Proposition 21)}$$

$$\Leftrightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2). \quad \square$$

Corollary 7. *Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ with closed one-to-many-distinguishability: $\mathcal{D}^1_m(\mathcal{F}_1) \subset \mathcal{D}^1_m(\mathcal{F}_2)$ if and only if $\mathcal{D}^1_1(\mathcal{F}_1) \subset \mathcal{D}^1_1(\mathcal{F}_2)$.*

Theorem 6. *Consider the learning frameworks ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.*

1. $\mathcal{D}^1_m(ILP_b) \subset \mathcal{D}^1_m(ILP_{sm}) \subset \mathcal{D}^1_m(ILP_{LAS}) \subset \mathcal{D}^1_m(ILP_{LOAS}) \subset \mathcal{D}^1_m(ILP^{context}_{LOAS})$
2. $\mathcal{D}^1_m(ILP_c) \subset \mathcal{D}^1_m(ILP_{LAS})$

Proof. Firstly, as shown in Example 12, $\mathcal{D}^1_m(ILP_b)$ is a strict subset of $\overline{\mathcal{D}^1_m(ILP_b)}$. Hence, by Theorem 5, $\mathcal{D}^1_m(ILP_b) \subset \overline{\mathcal{D}^1_m(ILP_{sm})}$; and hence as ILP_{sm} has closed one-to-many-distinguishability, $\mathcal{D}^1_m(ILP_b) \subset \mathcal{D}^1_m(ILP_{sm})$. The other results all follow from Corollary 7 and Proposition 23. □

Even if two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$ both have closed one-to-many-distinguishability, it might not be the case that their combination has closed one-to-many-distinguishability. Example 13 shows, for example, that this is not the case for ILP_{sm} and ILP_c. We define first what we mean by combination framework constructed from two given frameworks.

Definition 16. Given two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, the combination framework $comb(\mathcal{F}_1, \mathcal{F}_2)$ allows any task $\langle B, \langle 1, E_1 \rangle \rangle$, where $\langle B, E_1 \rangle$ is an $\mathcal{F}_1$ task, and any task $\langle B, \langle 2, E_2 \rangle \rangle$, where $\langle B, E_2 \rangle$ is an $\mathcal{F}_2$ task.

Given any $comb(\mathcal{F}_1, \mathcal{F}_2)$ task $T = \langle B, \langle x, E \rangle \rangle$: $ILP_{comb(\mathcal{F}_1, \mathcal{F}_2)}(T) = \begin{cases} ILP_{\mathcal{F}_1}(\langle B, E \rangle) & \text{if } x = 1 \\ ILP_{\mathcal{F}_2}(\langle B, E \rangle) & \text{if } x = 2 \end{cases}$

Example 13. Consider the frameworks ILP_{sm} and ILP_c (both of which have closed one-to-many-distinguishability). Consider the programs $B = \emptyset$, $H = \{0\{\mathtt{p}\}1.\}$, $H_1 = \emptyset$, $H_2 = \{0\{\mathtt{p}, \mathtt{q}\}1.\}$. $\langle B, H, H_1 \rangle \in \mathcal{D}^1_1(ILP_{sm})$ (using the task $\langle B, \langle \{\langle \{\mathtt{p}\}, \emptyset \rangle\} \rangle \rangle$), and $\langle B, H, H_2 \rangle \in \mathcal{D}^1_1(ILP_c)$ (using the task $\langle B, \langle \emptyset, \{\mathtt{q}\} \rangle \rangle$). This shows that both $\langle B, H, H_1 \rangle$ and $\langle B, H, H_2 \rangle$ are in $\mathcal{D}^1_1(ILP_{sm}) \cup \mathcal{D}^1_1(ILP_c)$. Hence by Definition 16 they must be in $\mathcal{D}^1_1(comb(ILP_{sm}, ILP_c))$. But using the distinguishability conditions proven in the previous section, it can be seen that neither framework can distinguish H from both H_1 and H_2. Therefore, $\langle B, H, \{H_1, H_2\} \rangle \notin \mathcal{D}^1_m(comb(ILP_{sm}, ILP_c))$. This also means that $\mathcal{D}^1_m(ILP_{sm}) \cup \mathcal{D}^1_m(ILP_c)$ is a strict subset of $\mathcal{D}^1_m(ILP_{LAS})$. This is because $\mathcal{D}^1_m(ILP_{LAS})$ contains both $\langle B, H, \{H_1\} \rangle$ and $\langle B, H, \{H_2\} \rangle$ (as it contains both $\mathcal{D}^1_m(ILP_{sm})$ and $\mathcal{D}^1_m(ILP_c)$) and has closed one-to-many-distinguishability, so must also contain $\langle B, H, \{H_1, H_2\} \rangle$.

5.3. The many-to-many-distinguishability class of a learning framework

So far, we have considered two main classes to define how general a learning framework is. Firstly, we discussed the one-to-one-distinguishability class, which is made up of tuples $\langle B, H, H' \rangle$ such that the framework can distinguish H from H' with respect to B. We showed that this has limitations and cannot separate ILP_b and ILP_{sm} even though ILP_b is clearly a special case of ILP_{sm}. This motivated upgrading the notion of a one-to-one-distinguishability class, changing the third

element of each tuple from a single hypothesis to a set of hypotheses to give the notion of a one-to-many-distinguishability class.

This naturally leads to the question of whether it is possible to upgrade generality classes by allowing the second element of the tuple to also be a set of hypotheses. Each tuple would then be of the form $\langle B, S_1, S_2 \rangle$, where B is a background knowledge, and S_1 and S_2 are sets of hypotheses. For each tuple in this new class, a framework would be required to have at least one task T with the background knowledge B such that every hypothesis in S_1 is an inductive solution of T, and no hypothesis in S_2 is an inductive solution of T. Definition 17 formalises this *many-to-many-distinguishability class*.

Definition 17. The *many-to-many-distinguishability class* of a learning framework $\mathcal{F}$ (denoted $\mathcal{D}^m_m(\mathcal{F})$) is the set of all tuples $\langle B, S_1, S_2 \rangle$, where B is a program and S_1 and S_2 are sets of hypotheses for which there is a task $T_{\mathcal{F}}$, with background knowledge B, such that $S_1 \subseteq ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $S_2 \cap ILP_{\mathcal{F}}(T_{\mathcal{F}})$. Given two frameworks, $\mathcal{F}_1$ and $\mathcal{F}_2$, we say that $\mathcal{F}_1$ is at least as (resp. more) $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ if and only if $\mathcal{D}^m_m(\mathcal{F}_2) \subseteq \mathcal{D}^m_m(\mathcal{F}_1)$ (resp. $\mathcal{D}^m_m(\mathcal{F}_2) \subset \mathcal{D}^m_m(\mathcal{F}_1)$).

We have already seen that for any two frameworks, $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2) \Rightarrow \mathcal{D}^1_1(\mathcal{F}_1) \subseteq \mathcal{D}^1_1(\mathcal{F}_2)$. We have also seen that for $\mathcal{D}^1_1$-generality and $\mathcal{D}^1_m$-generality, even if there is no corresponding strong reduction from $\mathcal{F}_2$ to $\mathcal{F}_1$ these subset relations are not necessarily strict. Proposition 24 and Corollary 8 show that $\mathcal{D}^m_m$-generality is equivalent to strong reductions.

Proposition 24. *For any two learning frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2 \Leftrightarrow \mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$.*

Proof.

1. Assume that $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. Let $\langle B, S_1, S_2 \rangle$ be an arbitrary element of $\mathcal{D}^m_m(\mathcal{F}_1)$. By definition of $\mathcal{D}^m_m(\mathcal{F}_1)$, there is a task $T_{\mathcal{F}_1}$ with background knowledge B such that $S_1 \subseteq ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and $S_2 \cap ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1}) = \emptyset$. Hence, as $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$, there is an $\mathcal{F}_2$ task $T_{\mathcal{F}_2}$ with background knowledge B such that $S_1 \subseteq ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2})$ and $S_2 \cap ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = \emptyset$. Hence $\langle B, S_1, S_2 \rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$.
2. Assume that $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$. Let $T_{\mathcal{F}_1}$ be an arbitrary $\mathcal{F}_1$ task. We must show that there is a $\mathcal{F}_2$ task with the same background knowledge and the same inductive solutions. Let B be the background knowledge of $T_{\mathcal{F}_1}$, $S_1 = ILP_{\mathcal{F}_1}(T_{\mathcal{F}_1})$ and S_2 be the (possibly infinite) set of ASP programs which are not in S_1. $\langle B, S_1, S_2 \rangle \in \mathcal{D}^m_m(\mathcal{F}_1)$; and hence, $\langle B, S_1, S_2 \rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$. Therefore, there must be at least one task $T_{\mathcal{F}_2}$ with the background knowledge B such that $ILP_{\mathcal{F}_2}(T_{\mathcal{F}_2}) = S_1$. Hence, $\mathcal{F}_1 \rightarrow_{sr} \mathcal{F}_2$. □

Corollary 8. *For any two learning frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ if and only if $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_{sr} \mathcal{F}_2$.*

Proposition 25. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$: $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2) \Rightarrow \mathcal{D}^1_m(\mathcal{F}_1) \subseteq \mathcal{D}^1_m(\mathcal{F}_2)$.*

Proof. Assume that $\mathcal{D}^m_m(\mathcal{F}_1) \subseteq \mathcal{D}^m_m(\mathcal{F}_2)$ and let $\langle B, H, S \rangle \in \mathcal{D}^1_m(\mathcal{F}_1)$. Then $\langle B, \{H\}, S \rangle \in \mathcal{D}^m_m(\mathcal{F}_1)$, and so $\langle B, \{H\}, S \rangle \in \mathcal{D}^m_m(\mathcal{F}_2)$. Hence, $\langle B, H, S \rangle \in \mathcal{D}^1_m(\mathcal{F}_2)$. □

Theorem 7 shows that one framework being more $\mathcal{D}^1_m$-general than another implies that it is also more $\mathcal{D}^m_m$-general if there is a strong reduction from the second framework to the first.

Theorem 7. *For any two frameworks $\mathcal{F}_1$ and $\mathcal{F}_2$, if $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ and $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$ then $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$.*

Proof. Assume that $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ and that $\mathcal{F}_2 \rightarrow_{sr} \mathcal{F}_1$. By Proposition 24, $\mathcal{F}_1$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_2$. It remains to show that $\mathcal{F}_2$ is not at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_1$. Assume for contradiction that $\mathcal{F}_2$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_1$. Then by Proposition 25, $\mathcal{F}_2$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_1$, contradicting the fact that $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$. □

Corollary 9. *Consider the learning frameworks ILP_b, ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$.*

1. $\mathcal{D}^m_m(ILP_b) \subset \mathcal{D}^m_m(ILP_{sm}) \subset \mathcal{D}^m_m(ILP_{LAS}) \subset \mathcal{D}^m_m(ILP_{LOAS}) \subset \mathcal{D}^m_m(ILP^{context}_{LOAS})$
2. $\mathcal{D}^m_m(ILP_c) \subset \mathcal{D}^m_m(ILP_{LAS})$

Proof. Each result follows directly from Theorem 6, Theorem 7 and Proposition 9. □

Note that although for each pair of frameworks discussed in this paper, one being more $\mathcal{D}^1_m$-general than another implies that it is also more $\mathcal{D}^m_m$-general, this result does not hold in general. Example 14 shows such a pair of frameworks.

Table 4
A summary of the relationships between the different measures of generality in this paper.

Property	Consequences of property
$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_1$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^m_m$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_1$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have equal $\mathcal{D}^1_m$-generality
$\mathcal{F}_1$ is more $\mathcal{D}^1_1$-general than $\mathcal{F}_2$	Either $\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$ or $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality
$\mathcal{F}_1$ is more $\mathcal{D}^1_m$-general than $\mathcal{F}_2$	1) Either $\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$ or $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is more $\mathcal{D}^m_m$-general than $\mathcal{F}_2$	1) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$	$\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ is at least as $\mathcal{D}^m_m$-general as $\mathcal{F}_2$	1) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_1$-general as $\mathcal{F}_2$ 2) $\mathcal{F}_1$ is at least as $\mathcal{D}^1_m$-general as $\mathcal{F}_2$
$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_1$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_m$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have different $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_1$-generality	1) $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality 2) $\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality
$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^1_m$-generality	$\mathcal{F}_1$ and $\mathcal{F}_2$ have incomparable $\mathcal{D}^m_m$-generality

Example 14. Consider a new learning framework ILP_d that takes as examples a pair of sets of atoms E^+ and E^- such that a hypothesis H is an inductive solution of a task if $B \cup H$ has exactly one answer set and this answer set contains all of the E^+'s and none of the E^-'s. The one-to-one-distinguishability class $\mathcal{D}^1_1(ILP_d) \subseteq \mathcal{D}^1_1(ILP_c)$. This can be seen as follows: assume that $\langle B, H, H'\rangle \in \mathcal{D}^1_1(ILP_d)$. Then there is a task $T_d = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H \in ILP_d(T_d)$ but $H' \notin ILP_d(T_d)$.

Case 1: $AS(B \cup H') = \emptyset$
Let $T_c = \langle B, \langle E^+, E^-\rangle\rangle$. As $B \cup H$ has exactly one answer set, and this answer set covers the examples, $H \in ILP_c(T_c)$. As $AS(B \cup H') = \emptyset$, $H' \notin ILP_c(T_c)$. Hence, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(ILP_c)$.

Case 2: $B \cup H'$ has exactly one answer set, and this answer set does not cover the examples.
Let $T_c = \langle B, \langle E^+, E^-\rangle\rangle$. As $B \cup H$ has exactly one answer set, and this answer set covers the examples, $H \in ILP_c(T_c)$. As $B \cup H'$ has an answer set that does not cover the examples, $H' \notin ILP_c(T_c)$. Hence, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(ILP_c)$.

Case 3: $B \cup H'$ has multiple answer sets.
There must be at least one answer set A^* of $B \cup H'$ that is not an answer set of $B \cup H$ (as $B \cup H$ only has one answer set). There must either be an atom $\mathtt{a} \in A^*$ that is not in the unique answer set of $B \cup H$, or an atom $\mathtt{a}$ that is not in A^*, but is in the unique answer set of $B \cup H$. In the first case, let $E^+_c = \emptyset$ and $E^-_c = \{\mathtt{a}\}$. In the second case, let $E^+_c = \{\mathtt{a}\}$ and $E^-_c = \emptyset$. Then let $T_c = \langle B, \langle E^+_c, E^-_c\rangle\rangle$. $H \in ILP_c(T_c)$ as the only answer set of $B \cup H$ covers the examples, whereas $H' \notin ILP_c(T_c)$ as $B \cup H'$ has at least one answer set that does not cover the examples. Hence, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(ILP_c)$.

In fact, $\mathcal{D}^1_1(ILP_d)$ is a strict subset of $\mathcal{D}^1_1(ILP_c)$ as $\mathcal{D}^1_1(ILP_d)$ has no elements $\langle B, H, H'\rangle$ where $B \cup H$ has multiple answer sets. As ILP_c is closed under one-to-many-distinguishability, and all one-to-many-distinguishability classes are subsets of their own closure, this means that ILP_c is more $\mathcal{D}^1_m$-general than ILP_d (by Theorem 5).

ILP_c is not, however, more $\mathcal{D}^m_m$-general than ILP_d. Take, for instance, the tuple $t = \langle \emptyset, \{\{\mathtt{heads.}\}, \{\mathtt{tails.}\}\}, \{\{\mathtt{1\{heads, tails\}1.}\}\}\rangle$. The empty set of examples are sufficient for ILP_d to distinguish both hypotheses containing facts from the choice rule (as the choice rule has multiple answer sets). However, there is no ILP_c task such that both facts are solutions, but the choice rule is not. Hence, $t \in \mathcal{D}^m_m(ILP_d)$ but $t \notin \mathcal{D}^m_m(ILP_c)$; and so, ILP_c is not at least as $\mathcal{D}^m_m$-general as ILP_d. In fact, as ILP_d is not as $\mathcal{D}^m_m$-general as ILP_c either, the two have incomparable $\mathcal{D}^m_m$-generalities.

Example 14 shows that $\mathcal{D}^m_m$-generality may not be able to compare two frameworks even when there is a clear $\mathcal{D}^1_m$-generality relation between the two. In the next section, we discuss relationships between, and the relative merits of using, each measure of generality.

5.4. Discussion

Table 4 summarises the relationships between the different measures of generality presented in this paper. It shows that equal one-to-one-distinguishability is weaker than equal one-to-many-distinguishability, which is weaker than equal many-to-many-distinguishability. This can be seen from the first section of the table, as equal many-to-many-distinguishability implies equal one-to-many-distinguishability, which implies equal one-to-one-distinguishability, but the converse implications do not hold in general. On the other hand different one-to-one-distinguishability is stronger than different one-to-many-distinguishability, which in turn is stronger than different many-to-many-distinguishability. This means that many-to-many-distinguishability (resp. one-to-many-distinguishability) will be able to "separate" frameworks that one-to-many-distinguishability (resp. one-to-one-distinguishability) can not; but, there are more frameworks that are incomparable under many-to-many-distinguishability (resp. one-to-many-distinguishability) than one-to-many-distinguishability (resp. one-to-one-distinguishability).

The different notions of generalities will never be inconsistent, in the sense that one will never say that $\mathcal{F}_1$ is more general than $\mathcal{F}_2$, while the other says that $\mathcal{F}_2$ is more general than $\mathcal{F}_1$. It is useful, however, to explain the tasks that the different measures of generality correspond to.

1. One-to-one-distinguishability describes how general a framework is at distinguishing one hypothesis from another.
2. One-to-many-distinguishability describes how general a framework is at the task of identifying one target hypothesis within a space of unwanted hypotheses.
3. Many-to-many-distinguishability describes how general a framework is for the task of identifying a set of target hypotheses – for any background knowledge B and set of hypotheses S, there is a task $T_{\mathcal{F}}$ with background knowledge B such that $ILP_{\mathcal{F}}(T_{\mathcal{F}}) = S$ if and only if $\langle B, S, \bar{S}\rangle \in \mathcal{D}^m_m(\mathcal{F})$, where $\bar{S}$ is the (infinite) set of hypotheses which are not in S.

In practice, as ILP usually addresses the task of finding a single target hypothesis from a space of other hypotheses, one-to-many-distinguishability is likely to be the most useful measure; however, one-to-one-distinguishability classes are useful for finding the one-to-many-distinguishability classes of frameworks, and many-to-many-distinguishability is interesting as a theoretical property.

5.4.1. More general learning frameworks

We have shown in this section that $ILP^{context}_{LOAS}$ is more general (under every measure) than any of the other tasks presented for learning under the answer set semantics. The obvious question is whether it is possible to go further and define more general learning tasks.

The most $\mathcal{D}^1_1$-general learning task possible would be able to distinguish between any two different ASP programs H_1 and H_2 with respect to any background knowledge B. This would require the learning task to distinguish between programs which are strongly equivalent, such as {`p. q:- p.`} and {`p:- q. q.`}. We would argue that this level of one-to-one-distinguishability is unnecessary as in ILP, we aim to learn programs whose output explains the examples. As two strongly equivalent programs will always have the same output, even when combined with additional programs providing "context", we can not see any reason for going further under $\mathcal{D}^1_1$-generality. As $ILP^{context}_{LOAS}$ has closed one-to-many-distinguishability, the same argument can be made for $\mathcal{D}^1_m$-generality.

One outstanding question is whether it is worth going any further under $\mathcal{D}^m_m$-generality. Note that it is possible to define the notion of the closure of many-to-many-distinguishability classes; however, none of the frameworks considered in this paper have closed many-to-many-distinguishability. It is unclear whether having closed many-to-many-distinguishability is a desirable property for a framework. Closed one-to-many-distinguishability means that a framework can distinguish a target hypothesis H from any set of hypotheses S such that it can distinguish H from each element of S: this means that the sets of examples that distinguish H from each element of S can be combined to form a single set of examples, ruling out each element of S. For a framework to have closed many-to-many-distinguishability, however, given two (or more) target hypotheses h_1, h_2 that can be distinguished from an undesirable hypothesis h_3, it would need to be able to find a task which distinguished both h_1 and h_2 from h_3. For example, as both $\langle \emptyset, \{$`heads.`$\}, \{1\{$`heads, tails`$\}1.\}\rangle$ and $\langle \emptyset, \{$`tails.`$\}, \{1\{$`heads, tails`$\}1.\}\rangle$ are in $\mathcal{D}^1_1(ILP_{LAS})$, for ILP_{LAS} to have closed many-to-many-distinguishability it would need to be able to find a task with an empty background knowledge that distinguishes both {`heads.`} and {`tails.`} from {`1{heads, tails}1.`}. It is difficult to imagine a scenario, however, where we should learn either the hypothesis that a coin is always `heads` or always `tails`, when the choice rule is not a desirable hypothesis.

5.4.2. The generality of noisy frameworks

As discussed in Section 4.4 some learning systems are able to solve tasks where examples are potentially noisy – in this case, not all examples should necessarily be covered, and there is a trade off between maximising coverage and not over-fitting the examples. One method, used by the XHAIL [42] and ILASP [32] systems is to penalise a hypothesis for each example that is not covered. Examples are given a positive integer penalty, which must be paid if the example is not covered.

The three measures of generality presented in this section could be extended to cover the noisy tasks. For instance, in the case of one-to-one-distinguishability we could define the "noisy" one-to-one-distinguishability class of a learning framework as the set of tuples $\langle B, H_1, H_2 \rangle$, for which there is a set of examples E such that $p(H_1, E) < p(H_2, E)$, where $p(H, E)$ is the total penalty paid by a hypothesis H (together with the background knowledge B) over the examples E. In fact, we now show that this extended notion of one-to-one-distinguishability class would be equivalent to the standard "non-noisy" one-to-one-distinguishability class.

As all penalties are positive, $p(H_1, E) < p(H_2, E)$ implies that $p(H_1, E') < p(H_2, E')$, where E' the set of all examples in E that are covered by H. Hence, there is a set of examples E' such that H_1 every example in E', but H_2 does not. This means that any $\langle B, H_1, H_2 \rangle$ that would be in the "noisy" one-to-one-distinguishability class is in the standard "non-noisy" one-to-one-distinguishability class. Similarly for any tuple $\langle B, H_1, H_2 \rangle$ in the standard one-to-one-distinguishability class, there is a set of examples E such that H_1 covers every example in E, and H_2 does not; hence $p(H_1, E) < p(H_2, E)$, and so $\langle B, H_1, H_2 \rangle$ would be in the "noisy" one-to-one-distinguishability class.

A similar argument holds for the one-to-many-distinguishability class; however, it is worth noting that it does not hold true for the many-to-many-distinguishability class. If we upgrade the many-to-many-distinguishability class in the same way then there are some tuples which are in the "noisy" many-to-many-distinguishability class for a framework, but not in the standard many-to-many-distinguishability class. Take for instance the example discussed in the previous section: $\langle \emptyset, \{\{\texttt{heads.}\}, \{\texttt{tails.}\}\}, \{1\{\texttt{heads}, \texttt{tails}\}1\texttt{.}\} \rangle$ is not in $\mathcal{D}^m_m(ILP_{LAS})$. However, if we consider the ILP_{LAS} examples $E = \langle \emptyset, \{\langle \{\texttt{heads}\}, \emptyset \rangle, \langle \{\texttt{tails}\}, \emptyset \rangle\} \rangle$, then $p(\{\texttt{heads.}\}, E) = 1$, $p(\{\texttt{tails.}\}, E) = 1$ and $p(\{1\{\texttt{heads}, \texttt{tails}\}1\texttt{.}\}, E) = 2$, meaning that $\langle \emptyset, \{\{\texttt{heads.}\}, \{\texttt{tails.}\}\}, \{1\{\texttt{heads}, \texttt{tails}\}1\texttt{.}\} \rangle$ would be in the "noisy" $\mathcal{D}^m_m(ILP_{LAS})$.

6. Related work

The complexity of ILP_b and ILP_c for verification and satisfiability were investigated in [28]. However, in that work, the results on satisfiability are for deciding whether or not a task has any solutions with no restrictions on the hypothesis space. This means that for both ILP_b and ILP_c deciding whether a task is satisfiable is equivalent to checking whether there is a model of B in which the examples are covered (a simpler decision problem). For this reason, the complexity of satisfiability for ILP_c in [28] was NP-complete, rather than Σ^p_2-complete. The complexities given of verification of a hypothesis given in [28] are also different from the ones in this paper, as they consider a different language for $B \cup H$. They consider disjunctive logic programs, whereas we investigated the complexity of learning programs without disjunction. The reason we chose not to consider disjunctive logic programs is that the systems available for ILP under the answer set semantics do not allow disjunction. For example, the systems for ILP_b [38,39] do not allow disjunction, and allowing disjunction would raise the complexity beyond the complexity of the tasks that are actually solved in practice by the existing systems.

As discussed in Section 5, the generality of a learning framework has been investigated before. In [45], the author defined generality in terms of reductions – one framework $\mathcal{F}_1$ was said to be more general than another framework $\mathcal{F}_2$ if and only if $\mathcal{F}_2 \rightarrow_r \mathcal{F}_1$ and $\mathcal{F}_1 \not\rightarrow_r \mathcal{F}_2$. We showed in Section 5 that our final notion of generality (many-to-many-distinguishability) coincides with a similar notion of strong reductions. The difference with strong reductions, as compared to the reductions in [45], is that strong reductions do not allow the background knowledge to be modified as part of the reduction. We showed in Example 8 that ILP_b reduces to ILP_c, but ILP_b does not strongly reduce to ILP_c. This is because any reduction from ILP_b to ILP_c must encode the examples in the background knowledge, which we would argue abuses the purpose of the background knowledge.

Aside from the differences in strong reductions and reductions, we discussed in Section 5 that one-to-many-distinguishability is more relevant when comparing the generalities of frameworks with respect to the task of finding a single hypothesis within a space of hypotheses. The reductions of [45] are closer to the notion of many-to-many-distinguishability, because they compare the set of solutions.

One key advantage to using our three notions of generality, rather than strong reductions or reductions, is for comparing the relative generalities of frameworks that do not strongly reduce to one another. For instance, we have seen that ILP_b and ILP_c are incomparable under $\mathcal{D}$-generality, but we can still reason that ILP_b is never $\mathcal{D}$-general enough to distinguish a hypothesis containing a constraint from the same hypothesis without the constraint. On the other hand, ILP_c may be $\mathcal{D}$-general enough to do so (for example, ILP_c can distinguish $\{\texttt{:- p.}\}$ from $\emptyset$ with respect to the background knowledge $\{0\{\texttt{p}\}1\texttt{.}\}$, with the task $\langle \{0\{\texttt{p}\}1\texttt{.}\}, \langle \emptyset, \{\texttt{p}\} \rangle \rangle$).

6.1. Other learning frameworks

Traditional ILP aims to learn Prolog style logic programs, often restricted to learning definite programs (with no negation as failure). For the shared subset of the languages learned by these ILP frameworks and the ASP frameworks (definite rules, not including lists), a definite learning task can be expressed as either a brave, or as a cautious task with the same examples as the definite task, and hypothesis space restricted to definite logic programs. As these frameworks do not support features such as choice rules or constraints or negation, and ASP frameworks do not support lists, a comparison of the generality is not very informative. A review of early efforts to extend ILP to learn normal logic programs was presented in [8]. The techniques discussed in [8] that operate under the stable model (or answer set) semantics require that all examples are covered in all stable models (or answer sets). This corresponds to cautious induction.

We have already discussed most of the other frameworks for ILP which work under the answer set semantics and shown in sections 4 and 5 how the complexity and generality of these frameworks compare to our own frameworks. In particular, we have shown that although the complexities of our three learning frameworks (ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$) are the same as cautious induction, there are some learning problems which can be represented in learning from answer sets that cannot be represented in either brave or cautious induction. One example of this is the learning of the rules of Sudoku. This is because brave induction cannot incentivise learning the constraints in the rules of Sudoku, and there are no useful examples that can be given to a cautious learner about the values of cells, since no cell has the same value in every valid Sudoku board.

Another early work on learning frameworks under the answer set semantics is *Induction from Answer Sets* [46]. In the paper, two learning algorithms IAS^{pos} and IAS^{neg} are presented. The task of IAS^{pos} is to learn a hypothesis that cautiously entails a set of examples. This corresponds to the task of cautious induction. IAS^{neg} on the other hand aims to find a hypothesis that does not cautiously entail each of a set of examples (i.e. there should be at least one answer set that does not contain each example). This is (in some sense reversed) brave induction. As shown in the paper, in general the IAS^{pos} and IAS^{neg} procedures are cannot be combined in general to compute a correct hypothesis.

Another framework, under the supported model semantics rather than the answer set semantics, is *Learning from Interpretation Transitions* (LFIT) [47]. In LFIT, the examples are pairs of interpretations $\langle I, J\rangle$ where J is the set of immediate consequences of I given $B \cup H$. In [24], we presented a mapping from any LFIT task to an $ILP^{context}_{LAS}$ task. This shows that the complexity of deciding both satisfiability and verification for LFIT is at most Σ^p_2-complete. The generality, on the other hand would be different to the tasks we have considered, since there are programs that are strongly equivalent under the answer sets semantics that have different supported models. Example 15 demonstrates a pair of such programs, and an example that learning from interpretations could use to distinguish between them.

Example 15. Consider the programs P_1 and P_2.

$$P_1 = \{\texttt{p :- p.}\}$$
$$P_2 = \emptyset$$

P_1 and P_2 are strongly equivalent under the answer set semantics. However, P_1 has the supported model $\{\texttt{p}\}$, whereas P_2 does not. LFIT can distinguish P_1 from P_2 (with respect to an empty background knowledge) with the example $\langle\{\texttt{p}\}, \{\texttt{p}\}\rangle$.

Example 15 shows that $ILP^{context}_{LOAS}$ has a distinguishability class which does not contain LFIT's distinguishability class. Conversely, LFIT cannot have a distinguishability class which contains $\mathcal{D}^1_1(ILP^{context}_{LOAS})$, as it cannot distinguish hypotheses containing weak constraints from the same hypotheses without the weak constraints. In fact, it does not even contain $\mathcal{D}^1_1(ILP_{LAS})$, as shown in Example 16.

Example 16. Consider the programs P_1 and P_2.

$$P_1 = \left\{\texttt{p.}\right\}$$
$$P_2 = \left\{\begin{array}{l}\texttt{p :- p.}\\ \texttt{p :- not p.}\end{array}\right\}$$

For both programs P_1 and P_2, the immediate consequences of any interpretation I is the set $\{\texttt{p}\}$. This means that no example could possibly distinguish P_1 from P_2 with respect to an empty background knowledge. Under the answer set semantics, however, P_1 has one answer set $\{\texttt{p}\}$, but P_2 has no answer sets. ILP_{LAS} can therefore distinguish P_1 from P_2 (with respect to the empty background knowledge), with the positive example $\langle\{\texttt{p}\}, \emptyset\rangle$.

ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ have different distinguishability classes to LFIT, but none is either more or less $\mathcal{D}^1_1$-general than LFIT. This is an interesting observation, as it demonstrates that even when two frameworks are incomparable under our measures of generality, we can still reason about their individual distinguishability classes and discuss hypotheses which one framework is powerful enough to distinguish between and another is not. For instance, ILP_{LAS} cannot distinguish between any two hypotheses that are strongly equivalent under the answer set semantics, but Example 15 shows that there are some cases where ILP_{LFIT} can.

6.2. Relation to probabilistic ILP

One of the advantages to learning ASP programs rather than Prolog programs is that ASP allows the modeling of non-determinism, either through unstratified negation or through choice rules. The latter can be seen in the coin examples throughout the paper, where we have shown that our ILP_{LAS} framework can learn that a coin can be either heads or tails, but not both.

Another method for achieving non-determinism in ILP is by adding probabilities. Probabilistic Inductive Logic Programming [48] is a combination of ILP with probabilistic reasoning. Its aim is to learn a logic program that is annotated with probabilities. The task of PILP is often divided into *structure learning*, where the underlying logic program is learned, and *parameter estimation* or *weight learning*, where the probabilities are learned. A key difference between ILP_{LAS} and PILP is that while both aim to learn programs which are non-deterministic, ILP_{LAS} aims to learn programs whose answer sets capture the set of possibilities, whereas PILP aims to learn a probability distribution over these possibilities.

Although there has been significant progress in the field of PILP [25,49–51,26] for learning annotated Prolog programs, PILP under the answer set semantics is still relatively young, and thus, there are few approaches. PrASP [52,53,27] considers the problem of weight learning, and in fact uses a similar example of learning about coins. This example illustrates the difference between weight learning and standard ILP. In ILP our task is to learn that there are exactly two possibilities (`heads` and `tails`); whereas in weight learning, the goal is to estimate probabilities of each possibility. PROBXHAIL [54] does attempt to combine structure learning and weight learning, but can only learn definite logic programs.

While the coin example used in this paper may be viewed as inherently probabilistic, there are situations in practice where we may wish to learn non-deterministic programs without considering probability; for instance, in policy learning. A policy may well permit many valid actions in a given scenario, and impose some constraints on these actions. The task is to learn a program whose answer sets reflect the set of valid options, rather than to estimate the probability of each action being taken.

7. Conclusion

In this paper we have investigated the complexity and generality of the state of the art frameworks for learning answer set programs. We have shown, for the two decision problems of verification that a hypothesis is an inductive solution of a task and deciding whether a given task is satisfiable, that brave induction (ILP_b) and induction of stable models (ILP_{sm}) have the same complexities, and that cautious induction (ILP_c), learning from answer sets (ILP_{LAS}), learning from ordered answer sets (ILP_{LOAS}) and context dependent learning from ordered answer sets ($ILP^{context}_{LOAS}$) also have the same complexities as each other, but higher than ILP_b and ILP_{sm}. Studying the complexity of decision problems for the learning frameworks is important, as it gives a sense of the price paid for choosing a particular framework. In contrast, generality is important, as it shows the advantages of choosing one framework over another, by specifying which hypotheses can be learned by each framework. When using ILP in practice, a trade off must be made between the complexity and generality of the framework. The generality classes presented in this paper can inform this decision, as it is likely to be influenced by the class of programs that must be learned.

We have introduced three new measures of generality ($\mathcal{D}^1_1$-generality, $\mathcal{D}^1_m$-generality and $\mathcal{D}^m_m$-generality), and shown that, both under our own measures of generality, and by using the concept of strong reductions, there is an ordering of the generalities of the frameworks considered in this paper. Although ILP_c, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ have the same computational complexities, ILP_c is less general than ILP_{LAS}, which is less general than ILP_{LOAS}, which is less general than $ILP^{context}_{LOAS}$, under each measure of generality. This ordering could have been seen using strong reductions, but our measures go further. They allow us to reason about why one framework is more $\mathcal{D}^1_1$-general than another, for example, by studying the class of tuples which are in one framework's distinguishability class, but not the others. They also allow us to discuss the generalities of frameworks which are incomparable under strong reductions; for example, there is no strong reduction from ILP_c to ILP_b, or from ILP_b to ILP_c. Our measures allow us to show, however, that ILP_b is not $\mathcal{D}^1_1$-general enough to distinguish a hypothesis containing a constraint from the same program without the constraint, but in some cases ILP_c is $\mathcal{D}^1_1$-general enough to do so.

In this paper, most of the results we have presented have addressed non-noisy learning frameworks. In general our ILASP systems do support noise, by allowing examples to be labeled with a penalty. In this case, ILASP searches for a hypothesis that minimises the sum $|H| + p(H, E)$, where $p(H, E)$ is the sum of all examples in a set E that are not covered by a hypothesis H. Such a hypothesis is called an optimal solution. For the two decision problems of verification and satisfiability, we have shown that the complexity results are unaffected. In current work we are investigating whether the complexities of the non-noisy frameworks and the noisy frameworks differ for the decision problem of verifying that a hypothesis is an optimal solution of a given task. In future work, we also hope to "upgrade" the propositional complexity results presented in this paper to apply to the learning of first order answer set programs.

Acknowledgements

This research is partially funded by the EPSRC project EP/K033522/1 "Privacy Dynamics" and by an EPSRC Doctoral Training Account EP/L504786/1.

Appendix A. Proofs

A.1. Proofs from Section 4

In this section, we present the proofs that were omitted from Section 4. In some of the proofs, we make use of predicate symbols to avoid continually introducing new atoms and to aid readability. As this section is restricted to propositional programs, any first order atom should be interpreted as a new propositional atom.

Proposition 1.

1. *Deciding both verification and satisfiability for ILP_b reduces polynomially to the corresponding ILP_{sm} decision problem.*
2. *Deciding both verification and satisfiability for ILP_{sm} reduces polynomially to the corresponding ILP_b decision problem.*

Proof.

1. Let $T_b = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ be any arbitrary ILP_b task.
   Consider the task $T_{sm} = \langle B, S_M, \langle\{\langle E^+, E^-\rangle\}\rangle\rangle$. $\forall H$, $H \in ILP_{sm}(T_{sm})$ if and only if $H \in ILP_b(T_b)$ and hence deciding verification for ILP_b reduces polynomially to deciding verification for ILP_{sm}. Similarly, as $ILP_{sm}(T_{sm}) = ILP_b(T_b)$, T_{sm} is satisfiable if and only if T_b is satisfiable; hence, deciding satisfiability for ILP_b reduces to deciding satisfiability for ILP_{sm}.
2. Let $T_{sm} = \langle B, S_M, \langle E\rangle\rangle$ be any arbitrary ILP_{sm} task.
   Let $n = |E|$, where $E = \{e_1, \ldots, e_n\}$.
   For each integer i from 1 to n, let f_i be a function which maps each atom $\mathtt{a}$ in $B \cup S_M$ to a new atom $\mathtt{a_i}$. We also extend this notation to work on sets of atoms and rules (and parts of rules) by replacing each atom $\mathtt{a}$ in the set or rule with $f_i(\mathtt{a})$.
   For each rule $R \in S_M$, define a new atom $\mathtt{in\_h_R}$.
   Consider the task $T_b = \langle B^b, S_M^b, \langle E^+, E^-\rangle\rangle$ where the components of the task are as follows ($append(R, \mathtt{a})$ is the rule R with the atom $\mathtt{a}$ appended to the body).

$$B^b = \left\{ \begin{matrix} f_1(R), \\ \ldots, \\ f_n(R) \end{matrix} \middle| R \in B \right\} \bigcup \left\{ \begin{matrix} append(f_1(R), \mathtt{in\_h_R}), \\ \ldots, \\ append(f_n(R), \mathtt{in\_h_R}) \end{matrix} \middle| R \in S_M \right\}$$

$$S_M^b = \left\{ \mathtt{in\_h_R} \,\middle|\, R \in S_M \right\}$$

$$E^+ = \left\{ \mathtt{f_i(inc)} \middle| \begin{matrix} e_i \in E, \\ e_i = \langle e_i^{inc}, e_i^{exc}\rangle, \\ \mathtt{inc} \in e_i^{inc} \end{matrix} \right\}$$

$$E^- = \left\{ \mathtt{f_i(exc)} \middle| \begin{matrix} e_i \in E, \\ e_i = \langle e_i^{inc}, e_i^{exc}\rangle, \\ \mathtt{exc} \in e_i^{exc} \end{matrix} \right\}$$

   For any solution H of T_b, define $g(H)$ to be $\{R \mid \mathtt{in\_h_R} \in H\}$. We now show that $ILP_{sm}(T_{sm}) = \{g(H') \mid H' \in ILP_b(T_b)\}$.
   Assume $H \in ILP_{sm}(T_{sm})$
   $\Leftrightarrow$ $H \subseteq S_M$ and $\forall e_i \in E, \exists A \in AS(B \cup H)$ such that A extends e_i.
   $\Leftrightarrow$ $H \subseteq S_M$ and $\forall e_i \in E, \exists A \in AS(f_i(B \cup H))$ such that A extends $\langle\{f_i(\mathtt{inc}) \mid \mathtt{inc} \in e_i^{inc}\}, \{f_i(\mathtt{exc}) \mid \mathtt{exc} \in e_i^{exc}\}\rangle$.
   $\Leftrightarrow$ $H \subseteq S_M$ and $\exists A \in AS(\{f_i(B \cup H) \mid 1 \leq i \leq n\})$ such that A extends $\langle E^+, E^-\rangle$ (as the atoms in each sub program are disjoint).
   $\Leftrightarrow$ $H \subseteq S_M$ and $\exists A \in AS(B^b \cup \{\mathtt{in\_h_R} \mid R \in H\})$ such that A extends $\langle E^+, E^-\rangle$ (by the splitting set theorem, using $\{\mathtt{in\_h_R} \mid R \in H\}$ as a splitting set).
   $\Leftrightarrow$ $\exists H' \subseteq S_M^b$ such that $g(H') = H$ and $\exists A \in AS(B^b \cup H')$ such that A extends $\langle E^+, E^-\rangle$
   $\Leftrightarrow$ $\exists H' \in ILP_b(T_b)$ such that $g(H') = H$
   $\Leftrightarrow$ $H \in \{g(H') \mid H' \in ILP_b(T_b)\}$
   $\forall H$, $H \in ILP_{sm}(T_{sm})$ if and only if $g(H) \in ILP_b(T_b)$ and hence deciding verification for ILP_{sm} reduces polynomially to deciding verification for ILP_b. Similarly, as $ILP_{sm}(T_{sm}) = \{g(H) \mid H \in ILP_b(T_b)\}$, T_{sm} is satisfiable if and only if T_b is satisfiable; hence, deciding satisfiability for ILP_b reduces to deciding satisfiability for ILP_{sm}. □

Proposition 2.

1. *Deciding both verification and satisfiability for ILP_c reduces polynomially to the corresponding ILP_{LAS} decision problem.*
2. *Deciding both verification and satisfiability for ILP_{LAS} reduces polynomially to the corresponding $ILP_{LOAS}^{context}$ decision problem.*

3. *Deciding both verification and satisfiability for $ILP^{context}_{LOAS}$ reduces polynomially to the corresponding ILP_{LOAS} decision problem.*
4. *Deciding both verification and satisfiability for ILP_{LOAS} reduces polynomially to the corresponding ILP^{s}_{LAS} decision problem.*

Proof.

1. Let T_c be any ILP_c task $\langle B, S_M, \langle E^+, E^-\rangle\rangle$.
Consider the ILP_{LAS} task $T_{LAS} = \langle B, S_M, \langle\{\langle\emptyset,\emptyset\rangle\}, \{\langle\emptyset, \{e^+\}\rangle \mid e^+ \in E^+\} \cup \{\langle\{e^-\},\emptyset\rangle \mid e^- \in E^-\}\rangle\rangle$.
By the definition of ILP_{LAS}, $H \in ILP_{LAS}$ if and only if $H \subseteq S_M$; $\exists A \in AS(B \cup H)$ such that A extends $\langle\emptyset,\emptyset\rangle$; $\forall e^+ \in E^+$, $\nexists A \in AS(B \cup H)$ such that A extends $\langle\emptyset, e^+\rangle$; and finally, $\forall e^- \in E^-$, $\nexists A \in AS(B \cup H)$ such that A extends $\langle e^-, \emptyset\rangle$.
This is true if and only if $H \subseteq S_M$, $B \cup H$ is satisfiable, $\forall e^+ \in E^+$ $\forall A \in AS(B \cup H)$, $e^+ \in A$ and $\forall e^- \in E^-$ $\forall A \in AS(B \cup H)$ $e^- \notin A$. This is the definition of H being a member of $ILP_c(T_c)$; hence, $ILP_c(T_c) = ILP_{LAS}(T_{LAS})$.
As $ILP_c(T_c) = ILP_{LAS}(T_{LAS})$, T_c is satisfiable if and only if T_{LAS} is satisfiable. Hence deciding the satisfiability of an ILP_c task can be reduced to deciding the satisfiability of an ILP_{LAS} task in polynomial time. Similarly, for any hypothesis H, $H \in ILP_c(T_c)$ if and only if $H \in ILP_{LAS}(T_{LAS})$; and so deciding verification for ILP_c reduces polynomially to deciding verification for ILP_{LAS}.
2. Let T_{LAS} be any ILP_{LAS} task $\langle B, S_M, \langle E^+, E^-\rangle\rangle$. Consider the ILP_{LOAS} task $T_{LOAS} = \langle B, S_M, \langle E^+, E^-, \emptyset, \emptyset\rangle\rangle$.
$ILP_{LAS}(T_{LAS}) = ILP_{LOAS}(T_{LOAS})$ and hence, T_{LAS} is satisfiable if and only if T_{LOAS} is satisfiable. Hence deciding the satisfiability of ILP_{LAS} reduces polynomially to deciding satisfiability for ILP_{LOAS}. Similarly, for any hypothesis H, $H \in ILP_{LAS}(T_{LAS})$ if and only if $H \in ILP_{LOAS}(T_{LOAS})$; and so deciding verification for ILP_{LAS} reduces polynomially to deciding verification for ILP_{LOAS}.
3. In [24], we presented a mapping from any $ILP^{context}_{LOAS}$ task to an ILP_{LOAS} task. The correctness of this mapping is proven in Theorem 1 of [24]. Given any $ILP^{context}_{LOAS}$ task, we can decide its satisfiability by using this mapping and checking the satisfiability of the resulting ILP_{LOAS} task. Similarly, given any hypothesis and $ILP^{context}_{LOAS}$ task, we can verify that the hypothesis is an inductive solution of the task by using the mapping. Hence, both satisfiability and verification for $ILP^{context}_{LOAS}$ reduce to satisfiability and verification (respectively), for ILP_{LOAS}.
4. We do this by translating an arbitrary ILP_{LOAS} task $T_{LOAS} = \langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$ to an ILP^{s}_{LAS} task. Before we do this, we define several new atoms used in our meta representation.
For $i \in \{1, 2\}$, let f_i be a function which maps each atom `a` in $B \cup S_M$ to a new atom $\texttt{a}_\texttt{i}$. We also extend this notation to work on sets of atoms and rules (and parts of rules) by replacing each atom `a` in the set or rule with $f_i(\texttt{a})$.
For each rule $R \in S_M$, define a new atom in_h_R.
For each weak constraint $W \in B \cup S_M$ let $id_1(W)$ and $id_2(W)$ be two new (propositional) atoms and let $wt(W)$ be the weight of W and $priority(W)$ be the priority level of W.
For any two terms $\texttt{t}_1$ and $\texttt{t}_2$, $dominates(\texttt{t}_1, \texttt{t}_2)$ is defined as below.

$$dominates(\texttt{t}_1, \texttt{t}_2) =$$

$$\left\{ \begin{array}{l} \texttt{dom\_lv(t}_1\texttt{, t}_2\texttt{, l):-} \\ \quad \texttt{\#sum\{id}_1\texttt{(W}_1\texttt{)} = \texttt{wt(W}_1\texttt{)}, \ldots, \texttt{id}_1\texttt{(W}_\texttt{n}\texttt{)} = \texttt{wt(W}_\texttt{n}\texttt{)}, \\ \quad\quad \texttt{id}_2\texttt{(W}_1\texttt{)} = -\texttt{wt(W}_1\texttt{)}, \ldots, \texttt{id}_2\texttt{(W}_\texttt{n}\texttt{)} = -\texttt{wt(W}_\texttt{n}\texttt{)}\texttt{\}} > 0\texttt{.} \\ \texttt{non\_dom\_lv(t}_1\texttt{, t}_2\texttt{, l):-} \\ \quad \texttt{\#sum\{id}_1\texttt{(W}_1\texttt{)} = \texttt{wt(W}_1\texttt{)}, \ldots, \texttt{id}_1\texttt{(W}_\texttt{n}\texttt{)} = \texttt{wt(W}_\texttt{n}\texttt{)}, \\ \quad\quad \texttt{id}_2\texttt{(W}_1\texttt{)} = -\texttt{wt(W}_1\texttt{)}, \ldots, \texttt{id}_2\texttt{(W}_\texttt{n}\texttt{)} = -\texttt{wt(W}_\texttt{n}\texttt{)}\texttt{\}} > 0\texttt{.} \\ \texttt{dom(t}_1\texttt{, t}_2\texttt{):- dom\_lv(t}_1\texttt{, t}_2\texttt{, l),} \\ \quad \texttt{not non\_bef(t}_1\texttt{, t}_2\texttt{, l).} \end{array} \middle| \begin{array}{l} l \text{ is a priority level in } B \cup S_M, \\ W_1 \ldots W_n \text{ are the weak} \\ \text{constraints in } B \cup S_M \text{ with level } l \end{array} \right\}$$

$$\cup \left\{ \texttt{non\_bef(t}_1\texttt{, t}_2\texttt{, l}_1\texttt{):- non\_dom\_lv(t}_1\texttt{, t}_2\texttt{, l}_2\texttt{).} \middle| \begin{array}{l} l_1, l_2 \text{ are levels in } B \cup S_M, \\ l_1 < l_2 \end{array} \right\}$$

Consider the task $T^s_{LAS} = \langle B', S'_M, \langle E^{+\prime}, E^{-\prime}\rangle\rangle$ where the individual components are defined below. For the positive and negative examples, it is a simple reification so that the examples relate to the new B' and S'_M. The brave orderings are mapped to positive examples which can only be covered by a hypothesis H if $B \cup H$ bravely respects the ordering example. For any hypothesis $H' \in S'_M$, the f_1 and f_2 in a single answer set of $B' \cup H'$ represent two answer sets of $B \cup H$ where H is the hypothesis in S_M corresponding to H'. Similarly, cautious orderings are mapped to negative examples, such that there is an answer set of $B' \cup H'$ which extends the example if there is a pair of answer sets of the corresponding $B \cup H$ which are ordered incorrectly (i.e. if $B \cup H$ does not cautiously respect the ordering).

$$B' = \{f_i(R) | R \in B, R \text{ is not a weak constraint}, i \in \{1, 2\}\}$$
$$\cup \{id_i(W) \texttt{:-} f_i(body(W))\texttt{.} \mid W \text{ is a weak constraint in } B, i \in \{1, 2\}\}$$
$$\cup \{append(f_i(R), in\_h_R) | R \in S_M, \text{ R is not a weak constraint }\}$$
$$\cup \{id_i(W) \texttt{:-} append(f_i(body(W)), in\_h_W)\texttt{.} \mid W \text{ is a weak constraint in } S_M, i \in \{1, 2\}\}$$

$$\cup dominates(1,2) \cup dominates(2,1)$$

$$\cup \{\texttt{dom:- dom(1,2). dom:- dom(2,1).}\}$$

$$S'_M = \{in\_h_R | R \in S_M\}$$

$$E^{+'} = \left\{ f_1(e^+) \middle| e^+ \in E^+ \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(1,2)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, > \rangle \in O^b \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(2,1)}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \geq \rangle \in O^b \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \neq \rangle \in O^b \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(2,1)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, < \rangle \in O^b \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(1,2)}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \leq \rangle \in O^b \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, = \rangle \in O^b \right\}$$

$$E^{-'} = \left\{ f_1(e^-) \middle| e^- \in E^- \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(1,2)}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, > \rangle \in O^c \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(2,1)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \geq \rangle \in O^c \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \neq \rangle \in O^c \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}), f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \cup \{\texttt{dom(2,1)}\} \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, < \rangle \in O^c \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom(1,2)}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, \leq \rangle \in O^c \right\}$$

$$\cup \left\{ \langle f_1(e_1^{inc}) \cup f_2(e_2^{inc}) \cup \{\texttt{dom}\}, f_1(e_1^{exc}) \cup f_2(e_2^{exc}) \rangle \middle| \langle \langle e_1^{inc}, e_1^{exc} \rangle, \langle e_2^{inc}, e_2^{exc} \rangle, = \rangle \in O^c \right\}$$

By using the splitting set theorem [35], it can be shown that for any $H' \in S'_M$:

$$AS(B' \cup H') = \left\{ A \middle| A \in AS \left(\begin{array}{l} f_1(A_1) \cup f_2(A_2) \\ \cup dominates(1,2) \cup dominates(2,1) \\ \cup \{\texttt{dom:- dom(1,2). dom:- dom(2,1).}\} \end{array} \right), A_1, A_2 \in AS(B \cup H) \right\}$$

Hence, as the rules in $dominates(\texttt{t}_1, \texttt{t}_2)$ describe exactly the behaviour of the weak constraints in $B \cup H$ for two answer sets (with $\texttt{dom}(\texttt{t}_1, \texttt{t}_2)$ being true if and only if the first answer set dominates the second):
$AS(B' \cup H') = \left\{ A' \middle| A = f_1(A_1) \cup f_2(A_2), A_1, A_2 \in AS(B \cup H) \right\}$, where A' is A augmented with $\texttt{dom}$ and $\texttt{dom(1,2)}$ when A_1 dominates A_2 and $\texttt{dom}$ and $\texttt{dom(2,1)}$ when A_2 dominates A_1.
For any hypothesis $H' \in S'_M$, let H be the corresponding hypothesis in S_M. The answer sets of $B' \cup H'$ correspond to the pairs of answer sets of $B \cup H$.
Each positive example $e^+ \in E^+$ is mapped to an example in $E^{+'}$ ensuring that at least one of the pairs of answer sets' first answer set covers e^+. Note that as each answer set of $B \cup H$ must be the first element of one of these pairs at least once, this is true if and only if $B \cup H$ covers each positive example.
Similarly each negative example $e^- \in E^-$ is mapped to an example in $E^{-'}$ ensuring that none of the pairs of answer sets' first answer set covers e^-. This is true if and only if $B \cup H$ does not cover any negative examples.
Each brave ordering example $\langle e_1, e_2, op \rangle \in O^b$ is mapped to a positive example ensuring that there is a pair of answer sets $\langle A_1, A_2 \rangle$ of $B \cup H$ such that A_1 covers e_1, A_2 covers e_2 and $\langle A_1, A_2, op \rangle \in ord(B \cup H)$. This is true if and only if $B \cup H$ bravely respects the ordering example.
Each cautious ordering example $\langle e_1, e_2, op \rangle \in O^c$ is mapped to a negative example ensuring that there is no pair of answer sets $\langle A_1, A_2 \rangle$ of $B \cup H$ such that A_1 covers e_1, A_2 covers e_2 and $\langle A_1, A_2, op \rangle \notin ord(B \cup H)$. This is true if and only if $B \cup H$ cautiously respects the ordering example.
Hence, H' is an inductive solution of $ILP^s_{LAS}(\langle B', S'_M, \langle E^{+'}, E^{-'} \rangle \rangle)$ if and only if H is an inductive solution of $ILP_{LOAS}(\langle B, S_M, \langle E^+, E^-, O^b, O^c \rangle \rangle)$.
This means that we can check the satisfiability of any ILP_{LOAS} task (and similarly verify a solution) by mapping the task to an ILP^s_{LAS} task as above. Note that this is a well defined ILP^s_{LAS} task as B contains only stratified aggregates.
As this mapping is polynomial in size of the original task, this means that verification and satisfiability for ILP_{LOAS} each reduces polynomially to the corresponding decision problem for ILP^s_{LAS}. □

Proposition 3. *Verifying whether a given H is an inductive solution of a general ILP_b task is NP-complete.*

Proof. Let T_b be any ILP_b task $\langle B, S_M, \langle E^+, E^-\rangle\rangle$. For any $H \subseteq S_M$, $H \in ILP_b(T_b)$ if and only if $B \cup H \cup \{\texttt{:- not e}^+\texttt{.} \mid e^+ \in E^+\} \cup \{\texttt{:- e}^-\texttt{.} \mid e^- \in E^-\}$ is satisfiable. As deciding the satisfiability of this program is NP-complete ($B \cup H$ contains only normal rules, choice rules and constraints), and the program can be constructed in polynomial time, this means that deciding verification for ILP_b is in NP.

It remains to show that deciding verification is NP-hard. We do this by showing that deciding satisfiability for any ASP program P containing normal rules, choice rules and constraints can be reduced polynomially to deciding verification for an ILP_b task. Consider the ILP_b task $T_b = \langle P, \emptyset, \langle \emptyset, \emptyset\rangle\rangle$. Let $H = \emptyset$. $H \in ILP_b(T_b)$ if and only if there is an answer set of $P \cup H$, and hence, if and only if P is satisfiable. □

Proposition 4. *Deciding the satisfiability of a general ILP_b task is NP-complete.*

Proof. First we will show that deciding the satisfiability of a general ILP_b task is in NP. We do this by mapping an arbitrary task $T = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ to an ASP program whose answer sets can be mapped to the solutions of T. This program will be satisfiable if and only if T is satisfiable and as the program is aggregate stratified, checking whether the program is satisfiable is in NP. Hence, if we can construct such a program then we will have proved that deciding satisfiability for ILP_b is in NP.

For each $R^i \in S_M$ we define a new atom $\texttt{in\_h}_{\texttt{R}^\texttt{i}}$. Also, let $meta(R^i)$ be the rule R^i with the additional atom $\texttt{in\_h}_{\texttt{R}^\texttt{i}}$ added to the body.

We define the meta encoding T_{meta} as follows:

$$T_{meta} = B \cup \{meta(R^i) \mid R^i \in S_M\} \cup \{\ 0\{\texttt{in\_h}_{\texttt{R}^1}, \ldots, \texttt{in\_h}_{\texttt{R}^{|\texttt{S}_\texttt{M}|}}\}|\texttt{S}_\texttt{M}|.\ \}$$
$$\cup \{\texttt{:- not e.} \mid e \in E^+\} \cup \{\texttt{:- e.} \mid e \in E^-\}$$

For any answer set A, let $\mathcal{M}^{-1}(A) = \{R^i \mid R^i \in S_M, \texttt{in\_h}_{\texttt{R}^\texttt{i}} \in A\}$.

$A \in AS(T_{meta})$ if and only if $(A \backslash \{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}) \in AS(B \cup \mathcal{M}^{-1}(A) \cup \{\texttt{:- not e.} \mid e \in E^+\} \cup \{\texttt{:- e.} \mid e \in E^-\})$. (This can be seen by using the splitting set theorem, with $\{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}$ as the splitting set.)

Hence $A \in AS(T_{meta})$ if and only if $\exists H \subseteq S_M$ such that $H = \mathcal{M}^{-1}(A)$, $(A \backslash \{\texttt{in\_h}_{\texttt{R}^\texttt{i}} \mid R^i \in S_M\}) \in AS(B \cup H)$ and A respects the examples.

Hence T_{meta} is satisfiable if and only if $\exists H \subseteq S_M$ such that $\exists A \in AS(B \cup H)$ such that A respects the examples. This is the case if and only if T is satisfiable.

It remains to show that deciding the satisfiability of a general ILP_b task is NP-hard. Deciding the satisfiability of a normal logic program is NP-hard, so demonstrating that a deciding the satisfiability of a normal program P can be mapped to a ILP_b task is sufficient.

Let P be any normal logic program. Let T be the ILP_b task $\langle P, \emptyset, \langle \emptyset, \emptyset\rangle\rangle$. T is satisfiable if and only if $\exists H \subseteq \emptyset$ such that $\exists A \in AS(P \cup H)$ such that $\emptyset \subseteq A$ and $A \cap \emptyset = \emptyset$. This is true if and only if P is satisfiable.

Hence, deciding the satisfiability of a general ILP_b task is NP-complete. □

Proposition 5. *Deciding verification for ILP^s_{LAS} is a member of DP.*

Proof. Checking whether H is an inductive solution of an ILP^s_{LAS} task $T = \langle B, S_M, \langle E^+, E^-\rangle\rangle$ can be achieved by mapping T to two aggregate stratified ASP programs P^+ and P^-, such that $H \in ILP_{LAS}(T)$ if and only if P^+ bravely entails an atom and P^- cautiously an atom.

1. Let n be the integer $|E^+|$.
   For any integer $i \in [1, n]$, let f_i be a function mapping the atoms a in $B \cup H$ to new atoms a_i. We extend the notation to allow f_i to act on ASP programs (substituting all atoms in the program).
   Let P^+ be the program:
   $$\left\{ f_i(B \cup H) \cup \left\{ \begin{array}{c} \texttt{covered(i):- f}_\texttt{i}(\texttt{e}^{\texttt{inc}}_1), \ldots, \texttt{f}_\texttt{i}(\texttt{e}^{\texttt{inc}}_\texttt{m}), \\ \texttt{not f}_\texttt{i}(\texttt{e}^{\texttt{exc}}_1), \ldots, \texttt{not f}_\texttt{i}(\texttt{e}^{\texttt{exc}}_\texttt{o})\texttt{.} \end{array} \right\} \middle| e^i = \langle \{e^{inc}_1, \ldots, e^{inc}_m\}, \{e^{exc}_1, \ldots, e^{exc}_o\}\rangle \in E^+ \right\}$$
   P^+ can be split into n sub programs $P_1 \ldots P_n$ where each program P_i contains the rules containing the atoms generated by f_i, plus the rule for $\texttt{covered(i)}$.
   As the atoms in each sub program are disjoint from the atoms of all other subprograms, $AS(P^+) = \{A_1 \cup \ldots \cup A_n \mid A_1 \in AS(P_1), \ldots, A_n \in AS(P_n)\}$. (This follows from applying the splitting set theorem $n-1$ times).
   For each $i \in [1, n]$, $P_i \models_b \texttt{covered(i)}$ if and only if $\exists A \in B \cup H$ such that A extends e_i (where e_i is the i^{th} positive example). Hence $P^+ \cup \{\texttt{covered:- covered(1)}, \ldots, \texttt{covered(n).}\} \models_b \texttt{covered}$ if and only if all the positive examples are covered. Therefore, checking whether all the positive examples are covered is in NP by Corollary 1.
   As checking that $H \subseteq S_M$ can be done in polynomial time, this means that checking both that $H \in S_M$ and all the positive examples are covered is in NP.

2. Let P^- be the program:

$$B \cup H \cup \{\texttt{covered:- not neg\_violated.}\} \cup \bigcup \left\{ \begin{array}{c} \texttt{neg\_violated:-}\ \texttt{e}_1^{\texttt{inc}}, \ldots, \texttt{e}_m^{\texttt{inc}}, \\ \texttt{not}\ \texttt{e}_1^{\texttt{exc}}, \ldots, \texttt{not}\ \texttt{e}_o^{\texttt{exc}}. \end{array} \middle| \langle \{e_1^{inc}, \ldots, e_m^{inc}\}, \{e_1^{exc}, \ldots, e_o^{exc}\} \rangle \in E^- \right\}$$

$P^- \models_c \texttt{covered}$ if and only if $\nexists A \in AS(B \cup H)$ such that $\exists e^- \in E^-$ such that A extends e^-. Hence checking that all negative examples are covered is in co-NP by Lemma 1.

Hence as $H \in ILP^s_{LAS}(T)$ if and only if $H \subseteq S_M$, all the positive examples are covered and all the negative examples are covered, verifying that $H \in ILP^s_{LAS}(T)$ can be reduced to checking one problem in NP and another problem in co-NP. This means that verifying a hypothesis is a solution of an ILP^s_{LAS} task is in DP. □

Proposition 6. *Deciding verification for ILP_c is DP-hard.*

Proof. To prove that verification that a hypothesis is a solution of an ILP_c task is DP-hard, we must prove that any problem in DP can be reduced to the verification task.

Let D be any arbitrary decision problem which is in DP.

By the definition of DP, this is the case if and only if there exist two decision problems D_1 and D_2 such that D_1 is in NP, D_2 is in co-NP and D returns yes if and only if both D_1 and D_2 return yes.

By Lemma 1 and Corollary 1, this is the case if and only if there are two programs P_1 and P_2 and two atoms $\texttt{a}_1$ and $\texttt{a}_2$ such that both $P_1 \models_b \texttt{a}_1$ and $P_2 \models_c \texttt{a}_2$ if and only if D returns yes. Without loss of generality we can assume that the atoms in P_1 (together with $\texttt{a}_1$) are disjoint from the atoms in P_2 (together with $\texttt{a}_2$).

Take T_c to be the ILP_c task $\langle B, S_M, \langle E^+, E^- \rangle\rangle$, where the individual components of the task are defined as follows:

- $B = P_1 \cup append(P_2, \texttt{a}_3) \cup \{\texttt{:- not a}_1\texttt{.}\ \ \texttt{0\{a}_3\texttt{\}1.}\ \ \texttt{a}_2\texttt{:- not a}_3\texttt{.}\}$ (where we assume $\texttt{a}_3$ to be a new atom and $append(P, \texttt{a})$ to add the atom $\texttt{a}$ to the body of all rules in P)
- $S_M = \emptyset$
- $E^+ = \{\texttt{a}_2\}$
- $E^- = \emptyset$

$\emptyset \in ILP_c(T_c)$ if and only if $(P_1 \cup \{\texttt{:- not a}_1\texttt{.}\})$ is satisfiable and $append(P_2, \texttt{a}_3) \cup \{\texttt{0\{a}_3\texttt{\}1.}\ \ \texttt{a}_2\texttt{:- not a}_3\texttt{.}\} \models_c \texttt{a}_2$. This is the case as the two subprograms $P_1 \cup \{\texttt{:- not a}_1\texttt{.}\}$ and $append(P_2, \texttt{a}_3) \cup \{\texttt{0\{a}_3\texttt{\}1.}\ \ \texttt{a}_2\texttt{:- not a}_3\texttt{.}\}$ are disjoint, and the latter is guaranteed to be satisfiable (it will always have the answer set $\{\texttt{a}_2\}$).

Hence $\emptyset \in ILP_c(T_c)$ if and only if $P_1 \models_b \texttt{a}_1$, and $P_2 \models_c \texttt{a}_2$. But this is the case if and only if D returns yes.

Hence any problem in DP can be reduced to verifying that a hypothesis is an inductive solution of an ILP_c task.

Hence verification for ILP_c is DP-hard. □

Proposition 7. *Deciding satisfiability for ILP^s_{LAS} is in Σ_2^P.*

Proof. Given an ILP^s_{LAS} task $T = \langle B, S_M, \langle E^+, E^- \rangle\rangle$, we show that a non-deterministic Turing Machine with access to an NP oracle could check satisfiability of T in polynomial time.

A non-deterministic Turing Machine can have $|S_M|$ choices to make (corresponding to selecting each rule as part of the hypothesis). This hypothesis can then be verified in polynomial time using an NP oracle, with two queries similar to the NP query and (the complement of) the co-NP query in Proposition 5, answering yes if and only if the first query returned yes and the second query returned no.

Such a Turing Machine would terminate answering yes if and only if the task is satisfiable (as there is a path through the Turing Machine which answers yes if and only if there is an hypothesis in S_M which is an inductive solution of the task).

Hence, deciding the existence of a solution for an ILP^s_{LAS} task is in Σ_2^P. □

Proposition 8. *Deciding satisfiability for ILP_c is Σ_2^P-hard.*

Proof. We show this by reducing a known Σ_2^P-complete problem (deciding the existence of an answer set for a ground disjunctive logic program [55]) to an ILP_c task.

Take any ground disjunctive logic program P. We will define an ILP_c task $T(P)$ which has a solution if and only if P has an answer set.

Let *Atoms* be the set of atoms in P. Let P' be the program constructed by replacing each negative literal $\texttt{not a}$ with the literal $\texttt{not in\_as(a)}$ (where *in_as* is a new predicate) and replacing each head $h_1 \vee \ldots \vee h_m$ with the counting aggregate $1\{h_1, \ldots, h_m\}m$ (empty heads are mapped to $1\{\}0$ - this is equivalent to $\bot$).

We define the learning task $T(P)$ as follows (`not_minimal` is a new atom):

$$B = P' \cup \left\{ \begin{array}{l} \texttt{:- a, not in\_as(a).} \\ \texttt{not\_minimal :- not a, in\_as(a).} \end{array} \middle| a \in Atoms \right\}$$

$$S_M = \{in\_as(a) \mid a \in Atoms\}$$

$$E^+ = \emptyset$$

$$E^- = \{\texttt{not\_minimal}\}$$

This task has a solution if there exists an $H \subseteq S_M$ such that $B \cup H$ is satisfiable and no answer set of $B \cup H$ contains `not_minimal`.

$$\Leftrightarrow \exists H \subseteq S_M \text{ st} \exists A \in AS \left(\left\{ \texttt{1\{h}_1, \ldots, \texttt{h}_\texttt{m}\texttt{\}m :- b}_1, \ldots, \texttt{b}_\texttt{n} \middle| \begin{array}{l} \texttt{1\{h}_1, \ldots, \texttt{h}_\texttt{m}\texttt{\}m :- b}_1, \ldots, \texttt{b}_\texttt{n}, \\ \quad \texttt{not in\_as(c}_1\texttt{)}, \ldots, \texttt{not in\_as(c}_\texttt{o}\texttt{).} \\ \qquad \{in\_as(c_1), \ldots, in\_as(c_o)\} \cap H = \emptyset \end{array} \in P', \right\} \right)$$

such that $A \subseteq \{a \mid in\_as(a) \in H\}$ and no answer set of $B \cup H$ contains `not_minimal`.

$$\Leftrightarrow \exists H \subseteq S_M \text{ st } \exists A \in AS \left(\left\{ \texttt{1\{h}_1, \ldots, \texttt{h}_\texttt{m}\texttt{\}m :- b}_1, \ldots, \texttt{b}_\texttt{n} \middle| \begin{array}{l} \texttt{1\{h}_1, \ldots, \texttt{h}_\texttt{m}\texttt{\}m :- b}_1, \ldots, \texttt{b}_\texttt{n}, \\ \quad \texttt{not in\_as(c}_1\texttt{)}, \ldots, \texttt{not in\_as(c}_\texttt{o}\texttt{).} \\ \qquad \{in\_as(c_1), \ldots, in\_as(c_o)\} \cap H = \emptyset \end{array} \in P', \right\} \right)$$

such that $A = \{a \mid in\_as(a) \in H\}$ and there is no strict subset of A which is also an answer set (or there would be an answer set of $B \cup H$ which contains `not_minimal`).

$\Leftrightarrow \exists H \subseteq S_M$ st $\{a \mid in\_as(a) \in H\}$ is a minimal model of

$$\left\{ \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{m} \texttt{ :- b}_1, \ldots, \texttt{b}_\texttt{n} \middle| \begin{array}{l} \texttt{1\{h}_1, \ldots, \texttt{h}_\texttt{m}\texttt{\}m :- b}_1, \ldots, \texttt{b}_\texttt{n}, \\ \quad \texttt{not in\_as(c}_1\texttt{)}, \ldots, \texttt{not in\_as(c}_\texttt{o}\texttt{).} \\ \qquad \{in\_as(c_1), \ldots, in\_as(c_o)\} \cap H = \emptyset \end{array} \in P', \right\}$$

$\Leftrightarrow \exists H \subseteq S_M$ st $\{a \mid in\_as(a) \in H\}$ is a minimal model of

$$\left\{ \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{m} \texttt{ :- b}_1, \ldots, \texttt{b}_\texttt{n} \middle| \begin{array}{l} \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{m} \texttt{ :- b}_1, \ldots, \texttt{b}_\texttt{n}, \\ \quad \texttt{not c}_1, \ldots, \texttt{not c}_\texttt{o}. \\ \{in\_as(c_1), \ldots, in\_as(c_o)\} \cap H = \emptyset \end{array} \in P, \right\}$$

$\Leftrightarrow \exists A \subseteq Atoms$ st A is a minimal model of

$$\left\{ \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{m} \texttt{ :- b}_1, \ldots, \texttt{b}_\texttt{n} \middle| \begin{array}{l} \texttt{h}_1 \vee \ldots \vee \texttt{h}_\texttt{m} \texttt{ :- b}_1, \ldots, \texttt{b}_\texttt{n}, \\ \quad \texttt{not c}_1, \ldots, \texttt{not c}_\texttt{o}. \\ \{c_1, \ldots, c_o\} \cap A = \emptyset \end{array} \in P, \right\}$$

$\Leftrightarrow \exists A \subseteq Atoms$ such that A is a minimal model of P^A

$\Leftrightarrow \exists A \subseteq Atoms$ such that A an answer set of P.

$\Leftrightarrow P$ is satisfiable.

Hence, deciding whether a disjunctive logic program is satisfiable can in general be mapped to the decision problem of checking the existence of solutions of a learning from answer sets task.

Therefore, deciding the existence of solutions of a ground ILP_c task is Σ_2^P-hard. □

A.2. Proofs from Section 5

Proposition 12. *For any programs P_1 and P_2, $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$ if and only if $AS(P_1) \subseteq AS(P_2)$.*

Proof.

- Assume that $AS(P_1) \subseteq AS(P_2)$

  $$\Leftrightarrow \forall A \in AS(P_1),\ A \in AS(P_2).$$

  Assume $c = \texttt{i}_1 \wedge \ldots \wedge \texttt{i}_\texttt{m}, \wedge \texttt{not e}_1, \ldots, \texttt{not e}_\texttt{n} \in \mathcal{E}_b(P_1)$. Then there must be an answer set A of P_1 which contains all of the `i`'s and none of the `e`'s. Hence, there is also such an answer set of P_2 which contains all of the `i`'s and none of the `e`'s. Hence, $c \in \mathcal{E}_b(P_2)$.

- Conversely, assume that $\mathcal{E}_b(P_1) \subseteq \mathcal{E}_b(P_2)$. Let $A \in AS(P_1)$, we must show that $A \in AS(P_2)$.
  Let $\mathcal{L}$ be the set $HB_{P_1} \cup HB_{P_2}$.
  As $A \in AS(P_1)$, $c = \mathtt{i}_1 \wedge \ldots \wedge \mathtt{i}_m, \wedge \mathtt{not}\ \mathtt{e}_1, \ldots, \mathtt{not}\ \mathtt{e}_n \in \mathcal{E}_b(P_1)$, where the $\mathtt{i}$'s are the set of atoms in A and the $\mathtt{e}$'s are the set of atoms in $\mathcal{L} \backslash A$. As $c \in \mathcal{E}_b(P_1)$, $c \in \mathcal{E}_b(P_2)$ and hence there is an answer set A' of P_2 which contains each $\mathtt{i} \in A$ but no atom $\mathtt{e} \in \mathcal{L} \backslash A$, and hence as $HB_{P_2} \subseteq \mathcal{L}$, $A' = A$. Hence $A \in AS(P_2)$. □

Proposition 13. $\mathcal{D}_1^1(ILP_b) = \left\{\langle B, H_1, H_2\rangle \middle|\ AS(B \cup H_1) \nsubseteq AS(B \cup H_2)\right\}$.

Proof. We prove this by showing that $\mathcal{D}_1^1(ILP_b) = \left\{\langle B, H_1, H_2\rangle \middle| \mathcal{E}_b(B \cup H_1) \nsubseteq \mathcal{E}_b(B \cup H_2)\right\}$, which is equal to the set $\left\{\langle B, H_1, H_2\rangle \middle| AS(B \cup H_1) \nsubseteq AS(B \cup H_2)\right\}$ by Proposition 12.

- We first show that if $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$ then there must be a conjunction in $\mathcal{E}_b(B \cup H_1)$ that is not in $\mathcal{E}_b(B \cup H_2)$.
  As $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$, there is an ILP_b task $T = \langle B, \langle\{\mathtt{i}_1, \ldots, \mathtt{i}_m\}, \{\mathtt{e}_1, \ldots, \mathtt{e}_n\}\rangle\rangle$ such that $H_1 \in ILP_b(T)$ and $H_2 \notin ILP_b(T)$.
  Hence, there must be an answer set of $B \cup H_1$ such that $\{\mathtt{i}_1, \ldots, \mathtt{i}_m\} \subseteq A$ and $\{\mathtt{e}_1, \ldots, \mathtt{e}_m\} \cap A = \emptyset$, but no such answer set of $B \cup H_2$.
  Hence the conjunction $c = \mathtt{i}_1 \wedge \ldots \wedge \mathtt{i}_m \wedge \mathtt{not}\ \mathtt{e}_1 \wedge \ldots \wedge \mathtt{not}\ \mathtt{e}_n \in \mathcal{E}_b(B \cup H_1)$ but $c \notin \mathcal{E}_b(B \cup H_2)$.
- Next we show that if there exists a conjunction $c = \mathtt{i}_1 \wedge \ldots \wedge \mathtt{i}_m \wedge \mathtt{not}\ \mathtt{e}_1 \wedge \ldots \wedge \mathtt{not}\ \mathtt{e}_n$ such that $c \in \mathcal{E}_b(B \cup H_1)$ but $c \notin \mathcal{E}_b(B \cup H_2)$, then $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$.
  Assume that there is such a conjunction c. Then $B \cup H_1$ has an answer set that extends $\langle\{\mathtt{i}_1, \ldots, \mathtt{i}_m\}, \{\mathtt{e}_1, \ldots, \mathtt{e}_n\}\rangle$ and $B \cup H_2$ does not. Hence $H_1 \in ILP_b(\langle B, \langle\{\mathtt{i}_1, \ldots, \mathtt{i}_m\}, \{\mathtt{e}_1, \ldots, \mathtt{e}_n\}\rangle\rangle)$ but $H_2 \notin ILP_b(\langle B, \langle\{\mathtt{i}_1, \ldots, \mathtt{i}_m\}, \{\mathtt{e}_1, \ldots, \mathtt{e}_n\}\rangle\rangle)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$. □

Proposition 14. $\mathcal{D}_1^1(ILP_b) = \mathcal{D}_1^1(ILP_{sm})$.

Proof.

- First we show that $\mathcal{D}_1^1(ILP_b) \subseteq \mathcal{D}_1^1(ILP_{sm})$. Assume $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$. Then there is a task $T_b = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_b(T_b)$ and $H_2 \notin ILP_b(T_b)$. Let $T_{sm} = \langle B, \{\langle E^+, E^-\rangle\}\rangle$. $H_1 \in ILP_{sm}(T_{sm})$ but $H_2 \notin ILP_{sm}(T_{sm})$. Hence, $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{sm})$.
- Next we show that $\mathcal{D}_1^1(ILP_b) \supseteq \mathcal{D}_1^1(ILP_{sm})$. Assume $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{sm})$. There must be a task $T_{sm} = \langle B, \{\langle E_1^+, E_1^-\rangle, \ldots, \langle E_n^+, E_n^-\rangle\}\rangle$ such that $H_1 \in ILP_{sm}(T_{sm})$ and $H_2 \notin ILP_{sm}(T_{sm})$. There must be at least one partial interpretation $\langle E_i^+, E_i^-\rangle$ such that there is an answer set A of $B \cup H_1$ such that $E_i^+ \subseteq A$ and $E_i^- \cap A = \emptyset$ and there is no such answer set of $B \cup H_2$. Hence, letting $T_b = \langle B, \langle E_i^+, E_i^-\rangle\rangle$, $H_1 \in ILP_b(T_b)$ but $H_2 \notin ILP_b(T_b)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_b)$. □

Proposition 15. $\mathcal{D}_1^1(ILP_c) = \left\{\langle B, H_1, H_2\rangle \middle| \begin{array}{c} AS(B \cup H_1) \neq \emptyset \wedge \\ (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_2) \nsubseteq \mathcal{E}_c(B \cup H_1)) \end{array}\right\}$.

Proof.

- First we show that for any $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_c)$, $AS(B \cup H_1) \neq \emptyset$ and either $AS(B \cup H_2) = \emptyset$ or $\mathcal{E}_c(B \cup H_1) \nsubseteq \mathcal{E}_c(B \cup H_2)$.
  Let $\langle B, H_1, H_2\rangle$ be an arbitrary element of $\mathcal{D}_1^1(ILP_c)$. As $H_1 \in ILP_c(T_c)$, $AS(B \cup H_1) \neq \emptyset$. Assume that $\mathcal{E}_c(B \cup H_1) \subseteq \mathcal{E}_c(B \cup H_2)$. We must show that $AS(B \cup H_2) = \emptyset$. As $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_c)$, $\exists T_c = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_c(T_c)$ and $H_2 \notin ILP_c(T_c)$.
  As $H_1 \in ILP_c(T_c), \forall A \in AS(B \cup H_1) : E^+ \subseteq A$ and $E^- \cap A \neq \emptyset$, hence the conjunction $E^+ \wedge \{\mathtt{not}\ e^- \mid e^- \in E^-\} \in \mathcal{E}_c(B \cup H_1)$; hence by our initial assumption that $\mathcal{E}_c(B \cup H_1) \subseteq \mathcal{E}_c(B \cup H_2)$, the conjunction is also in $\mathcal{E}_c(B \cup H_2)$; hence, $\forall A \in AS(B \cup H_2), E^+ \subseteq A$ and $E^- \cap A = \emptyset$. But as $H_2 \notin ILP_c(T_c)$ this means that $AS(B \cup H_2) = \emptyset$.
- We now show that for any B, H_1 and H_2, if $AS(B \cup H_1) \neq \emptyset \wedge (AS(B \cup H_2) = \emptyset \vee \mathcal{E}_c(B \cup H_1) \nsubseteq \mathcal{E}_c(B \cup H_2)$, then $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_c)$.
  Case 1: $AS(B \cup H_1) \neq \emptyset \wedge AS(B \cup H_2) = \emptyset$.
  Consider the task $T_c = \langle B, \langle\emptyset, \emptyset\rangle\rangle$. $H_1 \in ILP_c(T_c)$ as $AS(B \cup H_1) \neq \emptyset$ and $\forall A \in AS(B \cup H_1), \emptyset \subseteq A$ and $A \cap \emptyset = \emptyset$. $H_2 \notin ILP_c(T_c)$ as $AS(B \cup H_2) = \emptyset$.
  Case 2: $AS(B \cup H_1) \neq \emptyset \wedge \exists (c = \mathtt{i}_1 \wedge \ldots \wedge \mathtt{i}_m \wedge \mathtt{not}\ \mathtt{e}_1 \wedge \ldots \wedge \mathtt{e}_n) \in \mathcal{E}_c(B \cup H_1)$ such that $c \notin \mathcal{E}_c(B \cup H_2)$.
  Consider the task $T_c = \langle B, \langle\{i_1, \ldots, i_m\}, \{e_1, \ldots, e_n\}\rangle\rangle$. $H_1 \in ILP_c(T_c)$, but $H_2 \notin ILP_c(T_c)$. □

Proposition 16. $\mathcal{D}_1^1(ILP_{LAS}) = \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$

Proof.

- We first show that $\mathcal{D}_1^1(ILP_{LAS}) \subseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$. For any $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{LAS})$ there is an ILP_{LAS} task $T = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_{LAS}(T)$ and $H_2 \notin ILP_{LAS}(T)$.
  Case 1: $\exists e \in E^+$ such that $\forall A \in AS(B \cup H_2)$, A does not extend e.
  As $H_1 \in ILP_{LAS}(T), \exists A' \in AS(B \cup H_1)$ such that A' extends e. Hence as A' cannot be in $AS(B \cup H_2)$, $AS(B \cup H_1) \neq AS(B \cup H_2)$.
  Case 2: $\exists e \in E^-$, $\exists A \in AS(B \cup H_2)$ such that A extends e.
  As $H_1 \in ILP_{LAS}(T), \forall A' \in AS(B \cup H_1)$, A' does not extend e. Hence A cannot be in $AS(B \cup H_1)$ and so $AS(B \cup H_1) \neq AS(B \cup H_2)$.
- It remains to show that $\mathcal{D}_1^1(ILP_{LAS}) \supseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)\}$. Take B, H_1, H_2 to be any ASP programs such that $AS(B \cup H_1) \neq AS(B \cup H_2)$
  Let L be the set of atoms which appear in answer sets of $B \cup H_1$ and $B \cup H_2$.
  Case 1: $\exists A \in AS(B \cup H_1)$ such that $A \notin AS(B \cup H_2)$
  Let $e_A = \langle A, L \backslash A\rangle$. A is the only interpretation in $AS(B \cup H_1)$ or $AS(B \cup H_2)$ which extends e_A (as e_A is completely defined over the atoms in L). Hence, there is an answer set of $B \cup H_1$ which extends e_A, but no such answer set of $B \cup H_2$.
  Hence, $H_1 \in ILP_{LAS}(\langle B, \langle \{e_A\}, \emptyset\rangle\rangle)$, but $H_2 \notin ILP_{LAS}(\langle B, \langle \{e_A\}, \emptyset\rangle\rangle)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{LAS})$.
  Case 2: $\exists A \in AS(B \cup H_2)$ such that $A \notin AS(B \cup H_1)$
  Let $e_A = \langle A, L \backslash A\rangle$. A is the only interpretation in $AS(B \cup H_1)$ or $AS(B \cup H_2)$ which extends e_A. Hence, there is no answer set of $B \cup H_1$ which extends e_A, but there is such an answer set of $B \cup H_2$.
  Hence, $H_1 \in ILP_{LAS}(\langle B, \langle \emptyset, \{e_A\}\rangle\rangle)$, but $H_2 \notin ILP_{LAS}(\langle B, \langle \emptyset, \{e_A\}\rangle\rangle)$. So $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{LAS})$. □

Proposition 17. $\mathcal{D}_1^1(ILP_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \middle| \begin{array}{l} AS(B \cup H_1) \neq AS(B \cup H_2) \text{ or} \\ ord(B \cup H_1) \neq ord(B \cup H_2) \end{array} \right\}$.

Proof.

- We first show that $\mathcal{D}_1^1(ILP_{LOAS}) \subseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)\}$. For any $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{LOAS})$ there is an ILP_{LOAS} task $T = \langle B, \langle E^+, E^-, O^b, O^c\rangle\rangle$ such that $H_1 \in ILP_{LOAS}(T)$ and $H_2 \notin ILP_{LOAS}(T)$.
  Case 1: $\exists e \in E^+$ such that $\forall A \in AS(B \cup H_2)$, A does not extend e.
  As $H_1 \in ILP_{LOAS}(T), \exists A' \in AS(B \cup H_1)$ such that A' extends e. Hence as A' cannot be in $AS(B \cup H_2)$, $AS(B \cup H_1) \neq AS(B \cup H_2)$.
  Case 2: $\exists e \in E^-$, $\exists A \in AS(B \cup H_2)$ such that A extends e.
  As $H_1 \in ILP_{LOAS}(T), \forall A' \in AS(B \cup H_1)$, A' does not extend e. Hence, A cannot be in $AS(B \cup H_1)$ and so $AS(B \cup H_1) \neq AS(B \cup H_2)$.
  Case 3: $\exists \langle e_1, e_2, op\rangle \in O^b$ which is covered by H_1 but not H_2.
  Assume $AS(B \cup H_1) = AS(B \cup H_2)$. $\exists A_1, A_2 \in AS(B \cup H_1)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \in ord(B \cup H_1)$ as H_1 covers the ordering example. $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2)$ as H_2 does not cover the ordering example; and hence, $ord(B \cup H_1) \neq ord(B \cup H_2)$.
  Case 4: $\exists \langle e_1, e_2, op\rangle \in O^c$ which is covered by H_1 but not H_2.
  Assume $AS(B \cup H_1) = AS(B \cup H_2)$. $\exists A_1, A_2 \in AS(B \cup H_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2)$ as H_2 does not cover the ordering example. $\langle A_1, A_2, op\rangle \in ord(B \cup H_1)$ as H_1 does cover the ordering example; and hence, $ord(B \cup H_1) \neq ord(B \cup H_2)$.
  Hence, in all cases, either $AS(B \cup H_1) \neq AS(B \cup H_2$ or $ord(B \cup H_1) \neq ord(B \cup H_2)$.
- It remains to show that $\mathcal{D}_1^1(ILP_{LOAS}) \supseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)\}$. Take B, H_1, H_2 to be any ASP programs such that $AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)$.
  Let L be the set of literals which appear in answer sets of $B \cup H_1$ and $B \cup H_2$.
  Case 1: $AS(B \cup H_1) \neq AS(B \cup H_2)$
  $\langle B, H_1, H_2\rangle \in \mathcal{D}_1^1(ILP_{LAS})$ (by Proposition 16). Hence, there is an ILP_{LAS} task $T_{LAS} = \langle B, \langle E^+, E^-\rangle\rangle$ such that $H_1 \in ILP_{LAS}(T_{LAS})$ and $H_2 \notin ILP_{LAS}(T_{LAS})$. Let $T_{LOAS} = \langle B, \langle E^+, E^-, \emptyset, \emptyset\rangle\rangle$. $H_1 \in ILP_{LOAS}(T_{LOAS})$ and $H_2 \notin ILP_{LOAS}(T_{LOAS})$.
  Case 2: $AS(B \cup H_1) = AS(B \cup H_2)$ but $ord(B \cup H_1) \neq ord(B \cup H_2)$
  $\exists A_1, A_2 \in AS(B \cup H_1)$ (which is equal to $AS(B \cup H_2)$) such that there is a binary operator op such that $\langle A_1, A_2, op\rangle \in ord(B \cup H_1)$ but $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2)$.[6] Let $e_1 = \langle A_1, L \backslash A_1\rangle$ and $e_2 = \langle A_2, L \backslash A_2\rangle$ (where L is

[6] Note that there is no need to consider the case where there is a tuple $\langle A_1, A_2, op\rangle \in ord(B \cup H_2)$ such that $\langle A_1, A_2, op\rangle \notin ord(B \cup H_1)$, as in this case $\langle A_1, A_2, op^{-1}\rangle \in ord(B \cup H_1)$ and $\langle A_1, A_2, op^{-1}\rangle \notin ord(B \cup H_2)$ (where $<^{-1}$ is $\geq$, $>^{-1}$ is $\leq$, $=^{-1}$ is $\neq$, $\geq^{-1}$ is $<$, $\leq^{-1}$ is $>$ and $\neq^{-1}$ is $=$).

the set of atoms in the answer sets of $B \cup H_1$). Consider the ILP_{LOAS} task $T_{LOAS} = \langle B, \langle \{e_1, e_2\}, \emptyset, \{\langle e_1, e_2, op\rangle\}, \emptyset\rangle\rangle$. $H_1 \in ILP_{LOAS}(T_{LOAS})$ and $H_2 \notin ILP_{LOAS}(T_{LOAS})$.

Hence, in both cases $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP_{LOAS})$. □

Proposition 18. $\mathcal{D}^1_1(ILP^{context}_{LOAS}) = \left\{ \langle B, H_1, H_2\rangle \middle| \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \text{ such that } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$.

Proof.

- We first show that $\mathcal{D}^1_1(ILP^{context}_{LOAS}) \subseteq \left\{ \langle B, H_1, H_2\rangle \middle| \begin{array}{c} B \cup H_1 \not\equiv^s B \cup H_2 \text{ or} \\ \exists C \in \mathcal{ASP}^{ch} \text{ st } ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C) \end{array} \right\}$. For any $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP^{context}_{LOAS})$ there is an $ILP^{context}_{LOAS}$ task $T = \langle B, \langle E^+, E^-, O^b, O^c\rangle\rangle$ such that $H_1 \in ILP^{context}_{LOAS}(T)$ and $H_2 \notin ILP^{context}_{LOAS}(T)$.
  - **Case 1:** $\exists \langle e, C\rangle \in E^+$ such that $\forall A \in AS(B \cup H_2 \cup C)$, A does not extend e.
    As $H_1 \in ILP^{context}_{LOAS}(T)$, $\exists A' \in AS(B \cup H_1 \cup C)$ such that A' extends e. Hence, as A' cannot be in $AS(B \cup H_2 \cup C)$, $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$. Hence, $B \cup H_1 \not\equiv_s B \cup H_2$.
  - **Case 2:** $\exists \langle e, C\rangle \in E^-$, $\exists A \in AS(B \cup H_2 \cup C)$ such that A extends e.
    As $H_1 \in ILP^{context}_{LOAS}(T)$, $\forall A' \in AS(B \cup H_1 \cup C)$, A' does not extend e. Hence, A cannot be in $AS(B \cup H_1 \cup C)$ and so $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$. Hence, $B \cup H_1 \not\equiv_s B \cup H_2$.
  - **Case 3:** $\exists \langle \langle e_1, C_1\rangle, \langle e_2, C_2\rangle, op\rangle \in O^b$ which is covered by H_1 but not H_2.
    Assume that $B \cup H_1 \equiv_s B \cup H_2$.
    Let S be the set $AS(B \cup H_1 \cup C_1) \cup AS(B \cup H_1 \cup C_2)$ (which is equal to the set $AS(B \cup H_2 \cup C_1) \cup AS(B \cup H_2 \cup C_2)$ as $B \cup H_1 \equiv_s B \cup H_2$). $\exists A_1 \in AS(B \cup H_1 \cup C_1)$, $A_2 \in AS(B \cup H_1 \cup C_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \in ord(B \cup H_1, S)$ as H_1 covers the ordering example. $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2, S)$ as H_2 does not cover the ordering example.
    Let C be the $\mathcal{ASP}^{ch}$ program $append(C_1, \mathtt{a}_1) \cup append(C_2, \mathtt{a}_2) \cup \{1\{\mathtt{a}_1, \mathtt{a}_2\}1.\}$ (where $\mathtt{a}_1$ and $\mathtt{a}_2$ are new atoms and $append(P, \mathtt{a})$ appends the atom $\mathtt{a}$ to the body of each rule in P). $AS(B \cup H_1 \cup C) = \{A \cup \{\mathtt{a}_1\} \mid A \in AS(B \cup H_1 \cup C_1)\} \cup \{A \cup \{\mathtt{a}_2\} \mid A \in AS(B \cup H_1 \cup C_2)\}$, and hence, $t = \langle A_1 \cup \{\mathtt{a}_1\}, A_2 \cup \{\mathtt{a}_2\}, op\rangle \in ord(B \cup H_1 \cup C)$, but $t \notin ord(B \cup H_2 \cup C)$.
    Hence, $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.
  - **Case 4:** $\exists \langle e_1, e_2, op\rangle \in O^c$ which is covered by H_1 but not H_2.
    Assume that $B \cup H_1 \equiv_s B \cup H_2$.
    Let S be the set $AS(B \cup H_1 \cup C_1) \cup AS(B \cup H_1 \cup C_2)$ (which is equal to the set $AS(B \cup H_2 \cup C_1) \cup AS(B \cup H_2 \cup C_2)$ as $B \cup H_1 \equiv_s B \cup H_2$). $\exists A_1 \in AS(B \cup H_1 \cup C_1)$, $A_2 \in AS(B \cup H_1 \cup C_2)$ such that A_1 extends e_1, A_2 extends e_2 and $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2, S)$ as H_2 does not cover the ordering example. $\langle A_1, A_2, op\rangle \in ord(B \cup H_1, S)$ as H_1 does cover the ordering example.
    Let C be the $\mathcal{ASP}^{ch}$ program $append(C_1, \mathtt{a}_1) \cup append(C_2, \mathtt{a}_2) \cup \{1\{\mathtt{a}_1, \mathtt{a}_2\}1.\}$ (where $\mathtt{a}_1$ and $\mathtt{a}_2$ are new atoms and $append(P, \mathtt{a})$ appends the atom $\mathtt{a}$ to the body of each rule in P). $AS(B \cup H_1 \cup C) = \{A \cup \{\mathtt{a}_1\} \mid A \in AS(B \cup H_1 \cup C_1)\} \cup \{A \cup \{\mathtt{a}_2\} \mid A \in AS(B \cup H_1 \cup C_2)\}$, and hence, $t = \langle A_1 \cup \{\mathtt{a}_1\}, A_2 \cup \{\mathtt{a}_2\}, op\rangle \in ord(B \cup H_1 \cup C)$, but $t \notin ord(B \cup H_2 \cup C)$.
    Hence, $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.

  Hence, in all cases, either $B \cup H_1 \equiv_s B \cup H_2$ or $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$.
- It remains to show that $\mathcal{D}^1_1(ILP_{LOAS}) \supseteq \{\langle B, H_1, H_2\rangle | AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)\}$. Take B, H_1, H_2 to be any ASP programs such that $AS(B \cup H_1) \neq AS(B \cup H_2)$ or $ord(B \cup H_1) \neq ord(B \cup H_2)$.
  - **Case 1:** $B \cup H_1 \not\equiv_s B \cup H_2$
    There must be a program C such that $AS(B \cup H_1 \cup C) \neq AS(B \cup H_2 \cup C)$.
    - **Case i:** $\exists A \in AS(B \cup H_1 \cup C)$ such that $A \notin AS(B \cup H_2 \cup C)$.
      Let L be the set of atoms in the answer sets of $B \cup H_1 \cup C$ and $B \cup H_2 \cup C$ and let e_A be the partial interpretation $\langle A, L \backslash A\rangle$. Then $B \cup H_1 \cup C$ has an answer set that extends e_A, but $B \cup H_2 \cup C$ does not, and hence, $H_1 \in ILP^{context}_{LOAS}(\langle B, S_M, \langle \{\langle e_A, C\rangle\}, \emptyset, \emptyset, \emptyset\rangle\rangle)$ but H_2 is not.
    - **Case ii:** $\exists A \in AS(B \cup H_2 \cup C)$ such that $A \notin AS(B \cup H_1 \cup C)$.
      Let L be the set of atoms in the answer sets of $B \cup H_1 \cup C$ and $B \cup H_2 \cup C$ and let e_A be the partial interpretation $\langle A, L \backslash A\rangle$. Then $B \cup H_2 \cup C$ has an answer set that extends e_A, but $B \cup H_1 \cup C$ does not, and hence, $H_1 \in ILP^{context}_{LOAS}(\langle B, S_M, \langle \emptyset, \{\langle e_A, C\rangle\}, \emptyset, \emptyset\rangle\rangle)$ but H_2 is not.
  - **Case 2:** $B \cup H_1 \equiv_s B \cup H_2$ but $\exists C \in \mathcal{ASP}^{ch}$ such that $ord(B \cup H_1 \cup C) \neq ord(B \cup H_2 \cup C)$
    $\exists A_1, A_2 \in AS(B \cup H_1 \cup C)$ (which is equal to $AS(B \cup H_2 \cup C)$) such that there is a binary operator op such that $\langle A_1, A_2, op\rangle \in ord(B \cup H_1 \cup C)$ but $\langle A_1, A_2, op\rangle \notin ord(B \cup H_2 \cup C)$. Let $e_1 = \langle A_1, L \backslash A_1\rangle$ and $e_2 = \langle A_2, L \backslash A_2\rangle$ (where L is the set of atoms in the answer sets of $B \cup H_1 \cup C$). Consider the $ILP^{context}_{LOAS}$ task $T^{context}_{LOAS} = \langle B, \langle \{e_1, e_2\}, \emptyset, \{\langle \langle e_1, C\rangle, \langle e_2, C\rangle, op\rangle\}, \emptyset\rangle\rangle$. $H_1 \in ILP^{context}_{LOAS}(T^{context}_{LOAS})$ and $H_2 \notin ILP^{context}_{LOAS}(T^{context}_{LOAS})$.

  Hence, in both cases $\langle B, H_1, H_2\rangle \in \mathcal{D}^1_1(ILP^{context}_{LOAS})$. □

Proposition 19. *For any learning framework* $\mathcal{F}$, $\overline{\mathcal{D}^1_m(\mathcal{F})} = \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$.

Proof.

- We first show that $\overline{\mathcal{D}^1_m(\mathcal{F})} \subseteq \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$. Take an arbitrary $\langle B, H, S\rangle \in \overline{\mathcal{D}^1_m(\mathcal{F})}$. We must show that $\langle B, H, S\rangle \in \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$. To do this, we need to show that $\forall H' \in S$, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$.
  Take an arbitrary $H' \in S$. It remains to show that $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$. By definition of $\overline{\mathcal{D}^1_m(\mathcal{F})}$, there must be some subset $S' \subseteq S$ such that $H' \in S'$ and $\langle B, H, S'\rangle \in \mathcal{D}^1_m(\mathcal{F})$. Hence, $\exists T_{\mathcal{F}}$ such that $H \in ILP_{\mathcal{F}}(T_{\mathcal{F}})$ and $S' \cap ILP_{\mathcal{F}}(T_{\mathcal{F}}) = \emptyset$. Hence, as $H' \in S'$, $\langle B, H, H'\rangle \in \mathcal{D}^1_1(\mathcal{F})$.
- We now show that $\overline{\mathcal{D}^1_m(\mathcal{F})} \supseteq \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$. Take an arbitrary $\langle B, H, \{H_1, \ldots, H_n\}\rangle \in \left\{\langle B, H, \{H_1, \ldots, H_n\}\rangle \middle| \langle B, H, H_1\rangle, \ldots, \langle B, H, H_n\rangle \in \mathcal{D}^1_1(\mathcal{F})\right\}$. For each $i \in [1..n]$, $\langle B, H, H_i\rangle \in \mathcal{D}^1_1(\mathcal{F})$, and hence, $\langle B, H, \{H_i\}\rangle \in \mathcal{D}^1_m(\mathcal{F})$. Hence, by definition of $\overline{\mathcal{D}^1_m(\mathcal{F})}$, $\langle B, H, \{H_1, \ldots, H_n\}\rangle \in \overline{\mathcal{D}^1_m(\mathcal{F})}$. □

Proposition 20. *ILP_c, ILP_{sm}, ILP_{LAS}, ILP_{LOAS} and $ILP^{context}_{LOAS}$ all have closed one-to-many-distinguishability.*

Proof.

1. Consider any two ILP_c tasks, $T^1_c = \langle B, \langle E^+_1, E^-_1\rangle\rangle$ and $T^2_c = \langle B, \langle E^+_2, E^-_2\rangle\rangle$. Let $T^3_c = \langle B, \langle E^+_1 \cup E^+_2, E^-_1 \cup E^-_2\rangle\rangle$.
   $H \in ILP_c(T^1_c) \cap ILP_c(T^2_c)$ if and only if $AS(B \cup H)$ is non-empty and $\forall A \in AS(B \cup H)$: $E^+_1 \subseteq A$, $E^+_2 \subseteq A$, $E^-_1 \cap A = \emptyset$ and $E^-_2 \cap A = \emptyset$. This is the case if and only if $(E^+_1 \cup E^+_2) \subseteq A$, and $(E^-_1 \cup E^-_2) \cap A = \emptyset$ which holds if an only if $H \in ILP_c(T^3_c)$.
   Hence, by Lemma 2, ILP_c has closed one-to-many-distinguishability.
2. For any tasks $T^1_{sm} = \langle B, \{e^1_1, \ldots e^1_n\}\rangle$ and $T^2_{sm} = \langle B, \{e^2_1, \ldots e^2_m\}\rangle$, let $T^3_{sm} = \langle B, \{e^1_1, \ldots e^1_n, e^2_1, \ldots, e^2_m\}\rangle$. $ILP_{sm}(T^3_{sm}) = ILP_{sm}(T^1_{sm}) \cap ILP_{sm}(T^2_{sm})$.
   Hence, by Lemma 2, ILP_{sm} has closed one-to-many-distinguishability.
3. For any tasks $T^1_{LAS} = \langle B, \langle E^+_1, E^-_1\rangle\rangle$ and $T^2_{LAS} = \langle B, \langle E^+_2, E^-_2\rangle\rangle$, let $T^3_{LAS} = \langle B, \langle E^+_1 \cup E^+_2, E^-_1 \cup E^-_2\rangle\rangle$. $ILP_{LAS}(T^3_{LAS}) = ILP_{LAS}(T^1_{LAS}) \cap ILP_{LAS}(T^2_{LAS})$.
   Hence, by Lemma 2, ILP_{LAS} has closed one-to-many-distinguishability.
4. For any tasks $T^1_{LOAS} = \langle B, \langle E^+_1, E^-_1, O^b_1, O^c_1\rangle\rangle$ and $T^2_{LOAS} = \langle B, \langle E^+_2, E^-_2, O^b_2, O^c_2\rangle\rangle$, let $T^3_{LOAS} = \langle B, \langle E^+_1 \cup E^+_2, E^-_1 \cup E^-_2, O^b_1 \cup O^b_2, O^c_1 \cup O^c_2\rangle\rangle$. $ILP_{LOAS}(T^3_{LOAS}) = ILP_{LOAS}(T^1_{LOAS}) \cap ILP_{LOAS}(T^2_{LOAS})$.
   Hence, by Lemma 2, ILP_{LOAS} has closed one-to-many-distinguishability.
5. For any tasks $T1^{context}_{LOAS} = \langle B, \langle E^+_1, E^-_1, O^b_1, O^c_1\rangle\rangle$ and $T2^{context}_{LOAS} = \langle B, \langle E^+_2, E^-_2, O^b_2, O^c_2\rangle\rangle$, let $T3^{context}_{LOAS} = \langle B, \langle E^+_1 \cup E^+_2, E^-_1 \cup E^-_2, O^b_1 \cup O^b_2, O^c_1 \cup O^c_2\rangle\rangle$. $ILP^{context}_{LOAS}(T3^{context}_{LOAS}) = ILP^{context}_{LOAS}(T1^{context}_{LOAS}) \cap ILP^{context}_{LOAS}(T2^{context}_{LOAS})$.
   Hence, by Lemma 2, $ILP^{context}_{LOAS}$ has closed one-to-many-distinguishability. □

References

[1] S. Muggleton, Inductive logic programming, New Gener. Comput. 8 (4) (1991) 295–318.
[2] O. Ray, K. Broda, A. Russo, A hybrid abductive inductive proof procedure, Log. J. IGPL 12 (5) (2004) 371–397.
[3] S. Muggleton, L. De Raedt, D. Poole, I. Bratko, P. Flach, K. Inoue, A. Srinivasan, ILP turns 20, Mach. Learn. 86 (1) (2012) 3–23.
[4] S. Muggleton, D. Lin, Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, AAAI Press, 2013, pp. 1551–1557.
[5] A. Srinivasan, The Aleph Manual, Machine Learning at the Computing Laboratory, Oxford University, 2000.
[6] H. Blockeel, L. De Raedt, Top-down induction of first-order logical decision trees, Artif. Intell. 101 (1) (1998) 285–297.
[7] D. Corapi, A. Russo, E. Lupu, Inductive logic programming as abductive search, in: ICLP (Technical Communications), 2010, pp. 54–63.
[8] C. Sakama, Nonmonotomic inductive logic programming, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2001, pp. 62–80.
[9] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: ICLP/SLP, vol. 88, 1988, pp. 1070–1080.
[10] S. Kolb, Learning constraints and optimization criteria, in: Proceedings of the First Workshop on Declarative Learning Based Programming, 2016.
[11] C. Jordan, Ł. Kaiser, Machine learning with guarantees using descriptive complexity and smt solvers, preprint, arXiv:1609.02664.
[12] M. Sebag, C. Rouveirol, Constraint inductive logic programming.
[13] T. Eiter, G. Ianni, T. Krennwallner, Answer set programming: a primer, in: Reasoning Web. Semantic Technologies for Information Systems, Springer, 2009, pp. 40–110.
[14] E.T. Mueller, Commonsense Reasoning: An Event Calculus Based Approach, Morgan Kaufmann, 2014.
[15] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach, Cambridge University Press, 2014.
[16] E. Erdem, M. Gelfond, N. Leone, Applications of answer set programming, AI Mag. 37 (3) (2016) 53–68.
[17] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, M. Barry, An a-prolog decision support system for the space shuttle, in: International Symposium on Practical Aspects of Declarative Languages, Springer, 2001, pp. 169–183.
[18] F. Ricca, A. Dimasi, G. Grasso, S.M. Ielpa, S. Iiritano, M. Manna, N. Leone, A logic-based system for e-tourism, Fundam. Inform. 105 (1–2) (2010) 35–55.
[19] T. Soininen, I. Niemelä, Developing a declarative rule language for applications in product configuration, in: International Symposium on Practical Aspects of Declarative Languages, Springer, 1999, pp. 305–319.
[20] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gener. Comput. 4 (1) (1986) 67–95.

[21] N. Katzouris, A. Artikis, G. Paliouras, Incremental learning of event definitions with inductive logic programming, J. Mach. Learn. Res. 100 (2–3) (2015) 555–585.
[22] D. Athakravi, Inductive Logic Programming Using Bounded Hypothesis Space, Ph.D. thesis, Imperial College London, 2015.
[23] M. Law, A. Russo, K. Broda, Learning weak constraints in answer set programming, Theory Pract. Log. Program. 15 (4–5) (2015) 511–525.
[24] M. Law, A. Russo, K. Broda, Iterative learning of answer set programs from context dependent examples, Theory Pract. Log. Program. 16 (5–6) (2016) 834–848.
[25] L. De Raedt, A. Kimmig, H. Toivonen, Problog: a probabilistic prolog and its application in link discovery, in: IJCAI, vol. 7, 2007, pp. 2462–2467.
[26] F. Riguzzi, E. Bellodi, R. Zese, A history of probabilistic inductive logic programming, Front. Robot. AI 1 (2014) 6.
[27] M. Nickles, PrASP report, preprint, arXiv:1612.09591.
[28] C. Sakama, K. Inoue, Brave induction: a logical framework for learning from incomplete information, J. Mach. Learn. Res. 76 (1) (2009) 3–35.
[29] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, G. Terracina, The disjunctive datalog system DLV, in: Datalog Reloaded, Springer, 2011, pp. 282–301.
[30] R.P. Otero, Induction of stable models, in: Inductive Logic Programming, Springer, 2001, pp. 193–205.
[31] M. Law, A. Russo, K. Broda, Inductive learning of answer set programs, in: Logics in Artificial Intelligence (JELIA 2014), Springer, 2014.
[32] M. Law, A. Russo, K. Broda, The ILASP system for learning answer set programs, https://www.doc.ic.ac.uk/~ml1909/ILASP, 2015.
[33] M. Law, A. Russo, K. Broda, Simplified Reduct for Choice Rules in ASP, Tech. Rep. DTR2015-2, Imperial College of Science, Technology and Medicine, Department of Computing, 2015.
[34] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, F. Ricca, T. Schaub, ASP-Core-2 input language format, https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf, 2013.
[35] V. Lifschitz, H. Turner, Splitting a logic program, in: ICLP, vol. 94, 1994, pp. 23–37.
[36] C.H. Papadimitriou, Computational Complexity, John Wiley and Sons Ltd., 2003.
[37] L.J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1) (1976) 1–22.
[38] D. Corapi, A. Russo, E. Lupu, Inductive logic programming in answer set programming, in: Inductive Logic Programming, Springer, 2012, pp. 91–97.
[39] O. Ray, Nonmonotonic abductive inductive learning, J. Appl. Log. 7 (3) (2009) 329–340.
[40] S. Muggleton, Inverse entailment and progol, New Gener. Comput. 13 (3–4) (1995) 245–286.
[41] M. Law, A. Russo, K. Broda, Inductive learning of answer set programs v2.6.0.
[42] S. Bragaglia, O. Ray, Nonmonotonic learning in large biological networks, in: Inductive Logic Programming, Springer, 2015, pp. 33–48.
[43] D. Athakravi, D. Corapi, K. Broda, A. Russo, Learning through hypothesis refinement using answer set programming, in: International Conference on Inductive Logic Programming, Springer, 2013, pp. 31–46.
[44] W. Faber, G. Pfeifer, N. Leone, Semantics and complexity of recursive aggregates in answer set programming, Artif. Intell. 175 (1) (2011) 278–298.
[45] L. De Raedt, Logical settings for concept-learning, Artif. Intell. 95 (1) (1997) 187–201.
[46] C. Sakama, Induction from answer sets in nonmonotonic logic programs, ACM Trans. Comput. Log. 6 (2) (2005) 203–231.
[47] K. Inoue, T. Ribeiro, C. Sakama, Learning from interpretation transition, J. Mach. Learn. Res. 94 (1) (2014) 51–79.
[48] L. De Raedt, K. Kersting, Probabilistic inductive logic programming, in: International Conference on Algorithmic Learning Theory, Springer, 2004, pp. 19–36.
[49] L. De Raedt, I. Thon, Probabilistic rule learning, in: International Conference on Inductive Logic Programming, Springer, 2010, pp. 47–58.
[50] E. Bellodi, F. Riguzzi, Structure learning of probabilistic logic programs by searching the clause space, Theory Pract. Log. Program. 15 (02) (2015) 169–212.
[51] F. Riguzzi, E. Bellodi, R. Zese, G. Cota, E. Lamma, Scaling structure learning of probabilistic logic programs by MapReduce, in: European Conference on Artificial Intelligence, 2016.
[52] M. Nickles, A. Mileo, Probabilistic inductive logic programming based on answer set programming, preprint, arXiv:1405.0720.
[53] M. Nickles, A. Mileo, A system for probabilistic inductive answer set programming, in: International Conference on Scalable Uncertainty Management, Springer International Publishing, 2015, pp. 99–105.
[54] S. Dragiev, A. Russo, K. Broda, M. Law PROBXHAIL, An abductive-inductive algorithm for probabilistic inductive logic programming.
[55] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: propositional case, Ann. Math. Artif. Intell. 15 (3–4) (1995) 289–323.