Learning Weak Constraints in Answer Set Programming

Mark Law, Alessandra Russo, Krysia Broda
Department of Computing, Imperial College London, SW7 2AZ
(*e-mail:* {mark.law09, a.russo, k.broda}@imperial.ac.uk)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

This paper contributes to the area of non monotonic inductive logic programming by presenting a new learning framework that allows the learning of weak constraints in Answer Set Programming (ASP). The framework, called *Learning from Ordered Answer Sets*, generalises our previous work on learning ASP programs without weak constraints, by considering a new notion of examples as *ordered* pairs of partial answer sets that exemplify which answer sets of a learned hypothesis (together with a given background knowledge) are *preferred* to others. In this new learning task inductive solutions are searched within an hypothesis space of normal rules, choice rules, (hard) constraints and weak constraints. We propose a new algorithm, called ILASP2 which is sound and complete with respect to our new learning framework. We investigate its applicability to learning preferences in the setting of an interview scheduling problem and also demonstrate that when restricted to the task of learning ASP programs without weak constraints, ILASP2 can be much more efficient than our previously proposed system.

KEYWORDS: Non-monotonic Inductive Logic Programming, Preference Learning, Answer Set Programming

1 Introduction

Preference Learning has received much attention over the last decade from within the machine learning community. A popular approach to preference learning is *learning to rank* (Fürnkranz and Hüllermeier 2003; Geisler et al. 2001), where the goal is to learn to rank any two objects given some examples of pairwise preferences (indicating that one object is preferred to another). Many of these approaches use traditional machine learning tools such as neural networks (Geisler et al. 2001).

On the other hand, the field of Inductive Logic Programming (ILP) (Muggleton 1991) has seen significant advances in recent years, not only with the development of systems, such as (Ray et al. 2004; Kimber et al. 2009; Corapi et al. 2010; Muggleton et al. 2012; Muggleton and Lin 2013), but also the proposals of new frameworks for learning (Otero 2001; Sakama and Inoue 2009; Law et al. 2014). In most approaches to ILP, a learning task consists of a background knowledge B and sets of positive and negative examples. The task is then to find an hypothesis that, together with B, covers all the positive examples but none of the negative examples.

While in previous work ILP systems such as TILDE (Blockeel and De Raedt 1998) and Aleph (Srinivasan 2001) have been applied to preference learning (Dastani et al. 2001; Horváth 2012), this has addressed learning ratings, such as *good*, *poor* and *bad*, rather than learning full rankings over the examples. This is not expressive enough we want to use preferences to find an optimal solution as we may rate many objects as *good* when some are better than others.

Answer Set Programming (ASP), on the other hand, allows the expression of preferences through *weak constraints*. In a usual application of ASP, one would write a logic program which has many answer sets, each corresponding to a solution of the problem. The program can also contain weak constraints (or optimisation statements) which impose an ordering on the answer sets. Modern ASP solvers, such as clingo (Gebser et al. 2011), can then find the optimal answer sets, which correspond to the optimal solutions of the problem. For instance, in a scheduling problem, we could define an ASP program, whose answer sets correspond to timetables, and a set of weak constraints that define preferences over these timetables (see (Banbara et al. 2013) for an example application of ASP in timetabling).

In the paper, we propose a new learning framework, called *Learning from Ordered Answer Sets* (ILP_{LOAS}), that allows the learning of ASP programs with weak constraints. This new framework extends the notion of learning from answer sets that we proposed in (Law et al. 2014), where the learning of ASP programs was for programs without weak constraints and used only positive and negative examples of partial answer sets. In our new learning task ILP_{LOAS}, additional examples are defined as ordered pairs of partial answer sets and the language bias defines an hypothesis space of ASP programs that can be expressed using normal rules, choice rules, hard constraints and weak constraints. A new algorithm is presented and proved to be sound and complete with respect to the new learning task ILP_{LOAS}.

To demonstrate the applicability our framework, we will consider, as a running example, an interview timetabling problem and the task of learning, as weak constraints, academics' preferences for scheduling undergraduate interviews. For simplicity, we will assume an interview timetable consisting of 9 slots, three each on Monday, Tuesday and Wednesday. Answer Set solutions will consist of different assignments of academics to each of these interview slots. An academic might be more comfortable interviewing for one course than another, might prefer not to have many interviews on the same day, or might hold both of these preferences but regard the former as more important. Given ordered pairs of partial timetables, our learning approach is able to learn these preferences as weak constraints, which guarantees that the solutions will be ordered to reflect the preferences.

The paper is structured as follows. Our new learning framework, ILP_{LOAS} is presented in Section 3. It extends the notion of Learning from Answer Sets (Law et al. 2014) to the new task of learning weak constraints. The new learning algorithm *ILASP2* is described in Section 4, together with experimental results based on a scheduling example (Section 5). We also show that ILASP2 can have increased efficiency with respect to our previous system when learning programs without weak constraints. Discussion on related and future work concludes the paper.

2 Background

In this section we introduce the concepts needed in the paper. Given any atoms $h, h_1, \ldots, h_o, b_1, \ldots, b_n, c_1, \ldots, c_m$, $h \leftarrow b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m$ is called a *normal rule*, with h as the *head* and $b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m$ (collectively) as the *body* ("`not`" represents negation as failure); a rule $\leftarrow b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots,$ $\texttt{not}\ c_m$ is a *hard constraint*; a *choice rule* is a rule $l\{h_1, \ldots, h_o\}u \leftarrow b_1, \ldots, b_n,$ $\texttt{not}\ c_1, \ldots, \texttt{not}\ c_m$ (where l and u are integers) and its head is called an *aggregate*. A variable in a rule R is *safe* if it occurs in at least one positive literal in the body of R. A program P is assumed to be a finite set of normal rules, choice rules, and constraints. The Herbrand Base of P, denoted HB_P, is the set of all variable free (ground) atoms that can be formed from predicates and constants in P. The subsets of HB_P are called the (Hebrand) interpretations of P. A ground aggregate $l\{h_1, \ldots, h_o\}u$ is satisfied by an interpretation I if and only if $l \leq |A \cap \{h_1, \ldots, h_o\}| \leq u$. Any $I \subseteq HB_P$ is an *answer set* of P if it is the minimal model of the *reduct* P^I.

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 (Law et al. 2015b). Given a program P and an Herbrand interpretation $I \subseteq HB_P$, the reduct P^I is constructed from the grounding of 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 $\{h_1, \ldots, h_m\} \leftarrow b_1, \ldots, b_n$ with the set of rules $\{h_i \leftarrow b_1, \ldots, b_n \mid h_i \in I \cap \{h_1, \ldots, h_m\}\}$. Throughout the paper we denote the set of answer sets of a program P with $AS(P)$.

Unlike hard constraints in ASP, *weak constraints* do not affect what is, or is not, an answer set of a program P. Instead, they create an ordering over $AS(P)$ specifying which answer sets are "better" than others. The set of *optimal* (best) answer sets of P is denoted as $AS^*(P)$. A *weak constraint* is of the form $:\sim b_1, \ldots, b_n,$ $\texttt{not}\ c_1, \ldots, \texttt{not}\ c_m.[w@l, t_1, \ldots, t_o]$ where $b_1, \ldots, b_n, c_1, \ldots, c_m$ are atoms, w and l are terms specifying the *weight* and the *level*, and $t_1, \ldots, t_o$ are terms. A weak constraint W is *safe* if every variable occurring anywhere in W occurs in at least one positive literal in the body of W. At each *priority level* 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 l) whose bodies are true. The higher levels are minimised first. Terms specify which ground weak constraints should be considered unique. For any program P and $A \in AS(P)$, $weak(P, A)$ is the set of tuples $(w, l, t_1, \ldots, t_o)$ for which there is some $:\sim b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m.[w@l, t_1, \ldots, t_o]$ in the grounding of P such that A satisfies $b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m$. We now give the semantics for weak constraints (Calimeri et al. 2013).

Definition 1
For each level l, $P_A^l = \sum_{(w,l,t_1,\ldots,t_o) \in weak(P,A)} w$. 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 $P_{A_1}^l < P_{A_2}^l$ and $\forall m > l, P_{A_1}^m = P_{A_2}^m$. 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 consisting of `slot(m,1)`; `slot(m,2)`; `slot(t,1)`; `slot(t,2)`; and `0{assign(D,S)}1` $\leftarrow$ `slot(D,S)`, which assigns 0 to 4 slots in a schedule. Note that `slot(m,2)` represents slot 2 on Monday. Let W_1, W_2 and W_3 be the weak constraints $:\sim$ `assign(D,S).[1@1]`, $:\sim$ `assign(D,S).[1@1,D]` and $:\sim$ `assign(D,S).[1@1,D,S]` respectively. Applying each weak constraint to P gives as its optimal answer set the one in which no slots are assigned. The remaining answer sets are ordered in the following way: W_1 considers all schedules in which slots have been assigned, to be equally optimal, as there is only one unique set of terms $t_1, \ldots, t_n$ which is the empty set; W_2 minimises the number of days in which slots have been assigned, as there is one unique set of terms per day; and finally, W_3 minimises the number of assignments made, as each combination of day and slot has a unique set of terms.

In an ILP task, the hypothesis space is often characterised by mode declarations (Muggleton et al. 2012). A *mode bias* can be defined as a pair of sets of mode declarations $\langle M_h, M_b\rangle$, where M_h (resp. M_b) are the *head* (resp. *body*) declarations. Each mode declaration $m \in M_h$, or $m \in M_b$, is a literal whose abstracted arguments are either v or c. An atom is *compatible* with a mode declaration m if each instance of v in m is replaced by a variable, and every c by a constant. A rule of the form $h \leftarrow b_1, \ldots, b_n, \texttt{not}\ c_1, \ldots, \texttt{not}\ c_m$ is in the search space defined by M_h and M_b if and only if (i) h is empty, h is an atom compatible with some $m \in M_h$, or h is an aggregate $l\{h_1, \ldots, h_k\}u$ such that $0 \leq l \leq u \leq k$ and $\forall i \in [1,k]$ h_i is compatible with some $m \in M_h$; (ii) $\forall i \in [1,n]$, $\forall j \in [1,m]$ b_i and c_j are compatible with mode declarations in M_b; and finally (iii) all variables in the rule are safe. We denote the search space defined by a given mode bias (M_h, M_b) as $S_{LAS}(M_h, M_b)$.

In (Law et al. 2014), we presented a new learning task, *Learning from Answer Sets* (ILP_{LAS}) which used *partial interpretations* as its examples. A partial interpretation is a pair $e = \langle e^{inc}, e^{exc}\rangle$ of sets of ground atoms, called the *inclusions* and *exclusions*. An answer set A is said to *extend* a partial interpretation e if and only if $(e^{inc} \subseteq A) \wedge (e^{exc} \cap A = \emptyset)$.

Definition 2 (*from (Law et al. 2014)*)
A *Learning from Answer Sets* task is a tuple $T = \langle B, S_{LAS}(M_h, M_b), E^+, E^-\rangle$ where B is the background knowledge, $S_{LAS}(M_h, M_b)$ is the search space defined by a bias $\langle M_h, M_b\rangle$, E^+ and E^- are sets of partial interpretations called the positive and negative examples. An hypothesis H is an *inductive solution* of T, written $H \in ILP_{LAS}(T)$, if and only if $H \subseteq S_{LAS}(M_h, M_b)$; $\forall e^+ \in E^+\ \exists A \in AS(B \cup H)$ such that A extends e^+; and finally, $\forall e^- \in E^-\ \nexists A \in AS(B \cup H)$ such that A extends e^-.

The task of ILP is usually to find *optimal* hypotheses, where optimality is often defined by the number of literals in an hypothesis. In Learning from Answer Sets, aggregates are converted to disjunctions before the literals are counted, giving a higher "cost" for learning an aggregate; for example, $1\{p,q\}2$ is converted to $(p \wedge \texttt{not}\ q) \vee (q \wedge \texttt{not}\ p) \vee (p \wedge q)$ giving a length of 6. Learning from Answer Sets aims at learning ASP programs consisting of normal rules, choice rules and hard constraints. This paper extends this notion to a new learning task capable of learning weak constraints.

3 Learning from Ordered Answer Sets

To learn weak constraints we extend the notion of mode bias with two new sets of mode declarations: M_o and M_w. The former is used to specify what is allowed to appear in the body of a weak constraint, whereas the latter is used to specify what is allowed to appear as a weight. A positive integer l_{max} is also given to indicate the maximum number of levels that can occur in H.

Definition 3
A mode bias with ordering is a tuple $M = \langle M_h, M_b, M_o, M_w, l_{max}\rangle$, where M_h and M_b are respectively head and body declarations, M_o is a set of mode declarations for body literals in weak constraints, M_w is a set of integers and l_{max} is a positive integer. The search space S_M is the set of rules R that satisfy one of the conditions:

- $R \in S_{LAS}(M_h, M_b)$.
- R is a safe weak constraint $:\sim b_1, \ldots, b_i, \texttt{not}\ b_{i+1}, \ldots, \texttt{not}\ b_j\, .[w@l, t_1, \ldots, t_n]$ such that $\forall k \in [1, j]\ b_k$ is compatible with M_o; $t_1, \ldots, t_n$ is the set of terms in $b_1, \ldots, b_j$; $w \in M_w$, $l \in [0, l_{max})$.

Note that even if we were to extend the learning task in Definition 2 with this new notion of mode bias, such a task would never have as its optimal solution an hypothesis which contains a weak constraint. This is because a Learning from Answer Sets task has only examples of what is, or is not, an answer set. Any solution containing a weak constraint W will have the same answer sets without W, and would be shorter and therefore learned instead. We define now the notion of *ordering examples*.

Definition 4
An *ordering example* is a tuple $o = \langle e_1, e_2\rangle$ where e_1 and e_2 are partial interpretations. An ASP program P *bravely respects* o if and only if $\exists A_1, A_2 \in AS(P)$ such that A_1 extends e_1, A_2 extends e_2 and $A_1 \succ_P A_2$. P *cautiously respects* o if and only if for each $A_1, A_2 \in AS(P)$ such that $A_1 \neq A_2$, A_1 extends e_1 and A_2 extends e_2, it is the case that $A_1 \succ_P A_2$.

Example 2
Consider the partial interpretations $e_1 = \langle\{\texttt{assign(m,1)}, \texttt{assign(m,2)}\}, \{\texttt{assign(t,1)}, \texttt{assign(t,2)}\}\rangle$ and $e_2\langle\{\texttt{assign(m,1)}, \texttt{assign(t,1)}\}, \emptyset\rangle$. Let $o = \langle e_1, e_2\rangle$ be an ordering example and recall P and $W_1, \ldots, W_3$ from example 1. The only answer set of P that extends e_1 is M_1M_2 (where M_1M_2 denotes $\{\texttt{assign(m,1)}, \texttt{assign(m,2)}\}$), whereas the answer sets that extend e_2 are $M_1T_1, M_1M_2T_1, M_1M_2T_1T_2$ and $M_1T_1T_2$. $P \cup W_1$ does not bravely or cautiously respect o as it gives to all these answer sets the same optimality; $P \cup W_2$ both bravely and cautiously respects o, because each pair of answer sets extending the partial interpretations is ordered correctly (i.e. answer sets extending e_1 have slots allocated in only one day whereas all the answer sets extending e_2 have slots assigned in two days). Finally, $P \cup W_3$ bravely respects but does not cautiously respect o (i.e. the pair of answer sets M_1M_2 and M_1T_1 is such that $M_1M_2 \not\succ_P M_1T_1$).

We can now define the notion of *Learning from Ordered Answer Sets*.

Definition 5
A *Learning from Ordered Answer Sets* task is a tuple $T = \langle B, S_M, E^+, E^-, O^b, O^c \rangle$ where B is as ASP program, called the background knowledge, S_M is the search space defined by a mode bias with ordering M, E^+ and E^- are sets of partial interpretations called, respectively, positive and negative examples, and O^b and O^c are both sets of ordering examples over E^+. An hypothesis H is an *inductive solution* of T (written $H \in ILP_{LOAS}(T)$) if and only if:

1. $H' \in ILP_{LAS}(\langle B, S_{LAS}(M_h, M_b), E^+, E^- \rangle)$ where M_h and M_b in $S_{LAS}(M_h, M_b)$ are as in M and H' is the subset of H with no weak constraints.
2. $\forall o \in O^b$ $B \cup H$ bravely respects o
3. $\forall o \in O^c$ $B \cup H$ cautiously respects o

The notion of optimality is the same as in Learning from Answer Sets, with each weak constraint W counted as the number of literals in the body of W.

Example 3
Consider the learning from ordered answer sets task T with the following background knowledge:

$$B = \left\{ \begin{array}{l} \texttt{slot(m,1..3).slot(t,1..3).slot(w,1..3).} \\ \texttt{neq(1,2).neq(1,3).neq(2,1).neq(2,3).neq(3,1).neq(3,2).} \\ \texttt{neq(m,t).neq(m,w).neq(t,m).neq(t,w).neq(w,m).neq(w,t).} \\ \texttt{type(m,1,c1).type(m,2,c2).type(m,3,c2).type(t,1,c2).} \\ \texttt{type(t,2,c2).type(t,3,c2).type(w,1,c2).type(w,2,c1).type(w,3,c2).} \\ \texttt{0\{assign(X,Y)\}1:-slot(X,Y).} \end{array} \right\}$$

Using the notation from example 2, let T have the positive examples $e_1 = \langle \emptyset, M_2 M_3 T_1 T_3 W_1 W_2 \rangle$, $e_2 = \langle M_1 M_2, \emptyset \rangle$, $e_3 = \langle \emptyset, M_1 T_2 W_1 W_2 \rangle$, $e_4 = \langle T_1 T_2 T_3, \emptyset \rangle$, $e_5 = \langle M_2 M_3 T_1 T_2 T_3 W_1 W_3, \emptyset \rangle$; $e_6 = \langle M_1 W_1 W_3,\ M_2 M_3 T_1 T_2 T_3 W_2 \rangle$; two cautious orderings: $\langle e_1, e_2 \rangle$ and $\langle e_3, e_4 \rangle$; and one brave ordering: $\langle e_5, e_6 \rangle$. Consider S_M to be defined by the mode declarations: $M_h = M_b = \emptyset$; $M_o = \{$`assign(v,v)`, `neq(v,v)`, `type(v,c)`$\}$; $M_w = \{-1, 1\}$; and finally, $l_{max} = 2$. Then one optimal inductive solution of T is: $\{$`:~ assign(D,S1),assign(D,S2),neq(S1,S2).[1@1,D,S1,S2]; :~ assign(D,S), type(D,S,c1).[1@2,D,S]`$\}$. In fact, the other optimal solutions are equivalent hypotheses such as: $\{$`:~ assign(D,S1),assign(D,S2),neq(S1,S2).[1@1,D,S1,S2]; :~ assign(D,S), not type(D,S,c2).[1@2,D,S]`$\}$. Note that these hypotheses represent the academic's preferences described in the introduction. They express that the highest priority is to minimise the interviews for $c1$, and then to minimise the slots in any one day.

4 Algorithm

We now describe our new algorithm, *ILASP2*, capable of computing inductive solutions of any ILP_{LOAS} task, and present its soundness and completeness results with respect to the notion of Learning from Ordered Answer Sets task given in Definition 5. We omit the proofs of the theorems in this paper, but they can be found in full in (Law et al. 2015a). ILASP2 extends the concepts of *positive* and *violating* hypotheses, first introduced in our previous algorithm ILASP (Law et al.

2014), to cater for the new notion of ordering examples. An hypothesis is said to be *positive* if it covers all positive examples and bravely respects all the brave ordering examples. A positive hypothesis is defined to be *violating* if it covers at least one negative example or if it does not respect at least one of the cautious ordering examples. These two notions are formalised by Definitions 6 and 7.

Definition 6
Let $T = \langle B, S_M, E^+, E^-, O^b, O^c\rangle$ be an ILP_{LOAS} task. H is a *positive hypothesis*, written $H \in \mathcal{P}(T)$, if and only if $H \subseteq S_M$, $\forall e^+ \in E^+ \; \exists A \in AS(B \cup H)$ such that A extends e^+, and $\forall o \in O^b \; H \cup B$ bravely respects o.

Definition 7
A positive hypothesis H is a *violating hypothesis* of $T = \langle B, S_M, E^+, E^-, O^b, O^c\rangle$, written $H \in \mathcal{V}(T)$, iff at least one of the following cases is true:

- $\exists e^- \in E^-$ and $\exists A \in AS(B \cup H)$ such that A extends e^-. In this case we call A a violating interpretation of T and write $\langle H, A\rangle \in \mathcal{VI}(T)$.
- $\exists A_1, A_2 \in AS(B \cup H)$ and $\exists \langle e_1, e_2\rangle \in O^c$ such that A_1 extends e_1, A_2 extends e_2 and $A_1 \not\prec_P A_2$ with respect to $B \cup H$. In this case, we call $\langle A_1, A_2\rangle$ a *violating pair* of T and write $\langle H, \langle A_1, A_2\rangle\rangle \in \mathcal{VP}(T)$.

Example 4
Consider an ILP_{LOAS} task with B equal to P in Example 1 but with one additonal fact $\texttt{busy(1,1)}$; positive examples $e_1^+ = \langle\{\texttt{assign(t,1)}, \texttt{assign(t,2)}\}, \{\texttt{assign(m,2)}\}\rangle$ and $e_2^+ = \langle\{\texttt{assign(m,2)}, \texttt{assign(t,1)}\}, \emptyset\rangle$; one negative example $e_1^- = \langle\{\texttt{assign(m,1)}\}, \emptyset\rangle$; and one cautious ordering $\langle e_1^+, e_2^+\rangle$. S_M is unrestricted[1].

$H_1 = \emptyset \in \mathcal{P}(T)$ as e_1^+ and e_2^+ are covered and O^b is empty; however, $H_1 \in \mathcal{V}(T)$ for two reasons: firstly it has a violating interpretation $(\{\texttt{assign(m,1)}\})$; secondly it has a violating pair $(\langle\{\texttt{assign(t,1)},\texttt{assign(t,2)}\}, \{\texttt{assign(m,2)},\texttt{assign(t,1)}\}\rangle)$.

$H_2 = \{\leftarrow \texttt{busy(D,S)}, \texttt{assign(D,S)}\} \in \mathcal{P}(T)$. H_2 has no violating interpretations, but it has a violating pair $(\langle\{\texttt{assign(t,1)},\texttt{assign(t,2)}\}, \{\texttt{assign(m,2)}, \texttt{assign(t,1)}\}\rangle)$.

$H_3 = H_2 \cup \{:\sim \texttt{assign(D,S).[1@1,D]}\} \in \mathcal{P}(T)$. $H_3 \notin \mathcal{V}(T)$, as it has no violating interpretations and its weak constraints minimise the days assigned (so it cautiously respects the ordering example). It is, therefore, an inductive solution of the task.

One approach to computing the inductive solutions of an ILP_{LOAS} task would be to extend the original ILASP method with the above new notions of positive and violating hypotheses. Given a positive integer n, ILASP works by constructing an ASP meta representation of an ILP_{LAS} task T, called the *task program* T^n_{meta}, whose answer sets can be mapped back to the positive hypotheses of T of length n. T^n_{meta} can be augmented with an extra constraint so that its answer sets correspond exactly to the violating hypotheses of length n. ILASP first computes the violating hypotheses of length n, and then converts each of these to a constraint at the

[1] Hypotheses can be constructed from any predicate that appears in B and E. Note also that when we describe answer sets we omit the facts in B.

meta-level (ruling out that hypothesis). When T^n_{meta} is augmented with these new constraints, its answer sets correspond exactly to the positive hypotheses which were not computed the first time - the inductive solutions of length n.

The problem is that, in general, there can be many violating hypotheses which are shorter than the first inductive hypothesis and ILASP will compute all of them and add them into the task program as individual constraints. This scalability issue would be worsened if we were considering adding weak constraints to the search space. To overcome this, *ILASP*2 adopts a different strategy: it eliminates *classes* of hypothesis, i.e. hypotheses that are violating for the same "*reason*", namely they give rise to a particular violating interpretation or a particular violating pair of interpretations. The idea underlying the ILASP2 algorithm is to make use of two sets VI and VP which accumulate, respectively, violating interpretations and violating pairs of interpretations that are constructed during the search. We start initially with two empty sets VI and VP and continually compute the set of optimal *remaining hypotheses* which do not violate any of the *reasons* in VI or VP. If a computed hypothesis gives rise to a new violating interpretation then this interpretation is added to VI, if it gives rise to a violating pair of interpretations then this pair is added to VP. If no optimal remaining hypotheses are violating, then these hypotheses are the optimal inductive solutions of the task.

Definition 8
Let T be an ILP_{LOAS} task, VI and VP (resp.) be sets of violating interpretations and pairs of interpretations, and B be the background knowledge. Any $H \in \mathcal{P}(T)$ is a *remaining hypothesis* of T with respect to $VI \cup VP$ iff $VI \cap AS(B \cup H) = \emptyset$ and $\forall \langle I_1, I_2 \rangle \in VP$ if $I_1, I_2 \in AS(B \cup H)$ then $I_1 \succ_{B \cup H} I_2$. A remaining hypothesis H is a *remaining violating hypothesis* iff $\exists R$ such that $\langle H, R \rangle \in \mathcal{VI}(T) \cup \mathcal{VP}(T)$.

We use an ASP meta-level representation to solve our search for remaining hypotheses. As we rule out classes of hypothesis at the same time (rather than using one constraint per violating hypothesis), our meta-level representation is slightly more complex than that used in the original ILASP. Due to this added complexity, we define this representation in an appendix and give here the underling intuition.

The intuition of our meta encoding is that for a given task T, we construct an ASP program T_{meta} whose answer sets can be mapped back to the positive hypotheses of T. Given an answer set A of T_{meta} we write $\mathcal{M}^{-1}_{hyp}(A)$ to denote the hypothesis represented by A. Each positive hypothesis may be represented by many answer sets of T_{meta} but if this hypothesis gives rise to a violating interpretation, then at least one of these answer sets will contain a special atom v_i. If the hypothesis gives rise to a violating pair of interpretations then at least one of the answer sets of T_{meta} representing the hypothesis will contain a special atom $v\_p(t_1, t_2)$, where $\langle t_1, t_2 \rangle$ is a pair of identifiers corresponding to the cautious ordering example which is being violated. There is only one priority level in T_{meta} and the optimality of its answer sets is $2 * |H| + 1$ if the answer set does not contain the atom *violating* (*violating* is defined to be true if and only if v_i or at least one $v\_p(t_1, t_2)$ is true) and $2 * |H|$ if it does. This means that for any hypothesis H, the answer sets corresponding to H that do contain *violating* are preferred to those which do not.

We can use T_{meta} to find optimal positive hypotheses of T. If these positive solutions are violating, then the optimal answer sets will contain *violating*. We can then rule these hypotheses out. We can extract violating interpretations and violating pairs of interpretations from answer sets of T_{meta}, using the functions $\mathcal{M}^{-1}_{vi}$ and $\mathcal{M}^{-1}_{vp}$ respectively. Violating interpretations and violating pairs of interpretations are both called *violating reasons*. For any set of violating reasons $VR = VI \cup VP$, we then have a second meta encoding $VR_{meta}(T)$ which, when added to T_{meta}, rules out any hypotheses which are violating for a reason already in VR. This means that the answer sets of $T_{meta} \cup VR_{meta}(T)$ will represent the set of remaining hypotheses of T with respect to VR. These properties are guaranteed by Theorem 1.

Theorem 1
Given an ILP_{LOAS} task and a set of violating reasons VR, let AS be the set of optimal answer sets of $T_{meta} \cup VR_{meta}(T)$.

- If $\exists A \in AS$ such that $violating \in A$ then the set of optimal remaining violating hypotheses VH is non empty and is equal to the set $\{\mathcal{M}^{-1}_{hyp}(A) \mid A \in AS\}$.
- If no $A \in AS$ contains *violating*, then the set of optimal remaining hypotheses (none of which is violating) is equal to the set $\{\mathcal{M}^{-1}_{hyp}(A) \mid A \in AS\}$.

Algorithm 1 ILASP2

```
procedure ILASP2($T$)
    $VR = []$
    $solution = solve(T_{meta} \cup VR_{meta}(T))$
    while $solution \neq$ nil && $solution.optimality \% 2 == 0$ do
        $A = solution.answer\_set$
        if $v\_i \in A$ then
            $VR \mathrel{+}= \mathcal{M}^{-1}_{vi}(A)$
        else if $\exists t_1, t_2$ such that $v\_p(t_1, t_2) \in A$ then
            $VR \mathrel{+}= \mathcal{M}^{-1}_{vp}(A)$
        end if
        $solution = solve(T_{meta} \cup VR_{meta}(T))$
    end while
    return $\{\mathcal{M}^{-1}_{hyp}(A) \mid A \in AS^*(T_{meta} \cup VR_{meta}(T))\}$
end procedure
```

Algorithm 1 is the pseudo code of our algorithm ILASP2. It makes use of our meta encodings T_{meta} and $VR_{meta}(T)$. For any program P, $solve(P)$ is a function which, in the case that P is satisfiable, returns a pair consisting of an optimal answer set together and its optimality (there is only one priority level in our meta encoding, so this can be treated as an integer); if P is unsatisfiable then $solve(P)$ returns `nil`. While there are optimal remaining violating hypotheses, ILASP2 finds them and records the appropriate violating reasons. When the set of optimal remaining hypotheses contain no violating hypotheses then either the meta program will be unsatisfiable or the optimality of the optimal answer sets will be odd (by theorem 1), and so ILASP2 stops and returns the set of optimal remaining hypotheses.

Theorem 2 shows that ILASP2 is sound and complete with respect to the optimal inductive solutions of an ILP_{LOAS} task. This result relies on the termination of $ILASP2(T)$, which is guaranteed so long as $B \cup S_M$ grounds finitely.

Theorem 2
Let T be an ILP_{LOAS} task. If $ILASP2(T)$ terminates, then $ILASP2(T)$ returns the set of optimal solutions of $ILP_{LOAS}(T)$.

5 Experiments

Although there are benchmarks for ASP solvers (Denecker et al. 2009), there are no benchmarks for learning ASP programs. In (Law et al. 2014) we discussed the example of learning an ASP program with no weak constraints, representing the rules of sudoku. Using the examples from the paper and a small search space with only 283 rules, the original ILASP algorithm takes `486.2`s to solve the task. This is due to the scalability issues discussed in section 4 as there are 332437 violating hypotheses found before the first inductive solution. For the same task with ILASP2, there are only 9 violating reasons found before the first inductive solution, meaning that ILASP2 takes only `0.69`s to solve the task.

As this is the first work on learning weak constraints, there are no existing benchmarks suitable for testing our approach of learning from ordered answer sets. We have, therefore, further investigated the interview scheduling example discussed throughout the paper. Our experiments, in particular, test whether ILP_{LOAS} can successfully learn weak constraints from examples of brave and cautious orderings. For the purpose of presentation, we assume our hypothesis space, S_M, to be defined by the mode declarations: $M_h = M_b = \emptyset$; $M_o = \{\texttt{assign}(\texttt{v},\texttt{v}), \texttt{neq}(\texttt{v},\texttt{v}), \texttt{type}(\texttt{v},\texttt{c})\}$; $M_w = \{-1, 1\}$; and finally, $l_{max} = 2$. We place several restrictions on the search space in order to remove equivalent rules. The size of S_M is 184[2].

The learning task uses background knowledge B from Example 3. Although we have restricted our experiments to 3 day timetables with up to 20 ordering examples, $ILASP2$ has also been tested on 5 day timetables with over 100 examples. As S_M only contains weak constraints, for any $H \subseteq S_M$, $AS(B \cup H) = AS(B)$. The learning tasks described in these experiments therefore correspond to learning to rank the answer sets of B.

For each experiment we randomly selected 100 hypotheses with between 1 to 3 weak constraints from S_M, omitting hypotheses that ranked all answer sets equally. In our first experiment we investigated the effect of varying the number of examples, and in the second we investigated the effects of varying the fullness of the examples.

The only atoms that vary in B are the `assign`'s. As there are 9 different slots, there are 2^9 answer sets of B (and many more partial interpretations which are extended by these answer sets). We say an example partial interpretation is *full* if it specifies the truth value of all 9 `assign` atoms, otherwise we describe the *fullness* as the percentage of the 9 atoms which are specified. In both experiments (for each of the

[2] This means that our hypotheses can be any subset of these 184 rules, so even considering only hypotheses with up to 3 rules this gives over a million different hypotheses.

100 target hypotheses H_T), we generated ordered pairs of partial interpretations $o = \langle e_1, e_2 \rangle$ such that o was bravely respected. If o was also cautiously respected, then it was given as a cautious example (otherwise it was used as a brave example).

In both experiments, we tested our approach 20 times for each target hypothesis H_T. Each time, we used *ILASP*2 to learn an hypothesis H_L which covered all examples. We then calculated the accuracy of H_L in predicting the pairwise ordering of answer sets in B (for each pair of answer sets $A_1, A_2 \in AS(B)$ we tested whether H_T and H_L agreed on the preference between them).

In our first experiment we investigated the effect of varying the number of examples from 0 to 20. The examples were of random fullness with between 5 to 9 `assign` atoms specified. Figure 1(a) shows the average predictive accuracy. Each point on the graph corresponds to 2000 learning tasks (100 target hypotheses with 20 different sets of examples). The error bars on the graph show the standard error. The results show that our method achieves 90% accuracy for this experiment with around 10 or more random examples.

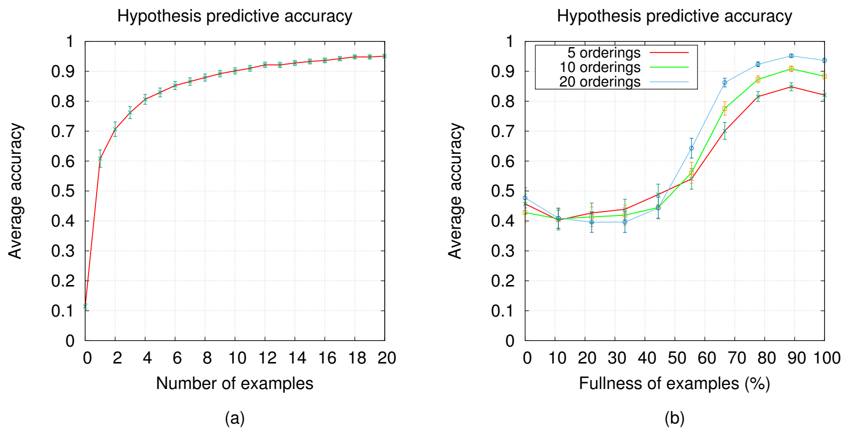

Fig. 1: (a) Accuracy with varying numbers of examples; (b) Accuracy with varying fullness of examples

For our second experiment we again tested our approach on 100 randomly generated hypotheses with 20 different sets of randomly generated examples. This time, however, we have kept the number of examples fixed at 5, 10 and 20 and varied the fullness of the examples. Results are shown in Figure 1(b). The graph shows that examples are only useful if they are more than 50% full. One interesting point to note is that the peak performance is with examples of around 90% fullness. This is because cautious ordering examples are actually more useful if they are less full (as there are more pairs which extend them); however, orderings are less likely to be cautiously respected when they are less full.

6 Related Work

In (Law et al. 2014) we showed that any of the learning tasks in (Corapi et al. 2012; Ray 2009; Sakama and Inoue 2009; Otero 2001) could be expressed by ILP_{LAS} and computed by *ILASP*. As any ILP_{LAS} task can be (trivially) mapped into an ILP_{LOAS} (i.e. $O^b = \emptyset$ and $O^c = \emptyset$), ILP_{LOAS} inherits this property. None of the previous learning tasks (including ILP_{LAS}), however, can construct examples which incentivise the learning of an hypothesis containing a weak constraint. This is because they can only give examples of what should (or shouldn't) be an answer set of $B \cup H$. In addition, ILP_{LOAS} inherits the capability of ILP_{LAS} of supporting *predicate invention*, allowing new concepts to be invented whilst learning.

The ILASP2 algorithm is an extension of the original ILASP algorithm in (Law et al. 2014). It extends the concepts of positive and violating hypothesis to cover learning weak constraints (which was not possible in ILASP). For the simpler learning from answer set tasks, ILASP2 is more efficient than ILASP. As discussed in section 4, the ILASP algorithm had some scalability issues when there is a large number of violating hypotheses. We have shown in section 5 that ILASP2 can eliminate violating reasons rather than single violating hypotheses and therefore be much more efficient.

Also related to our work are existing approaches for *learning to rank*. These use non logic-based machine learning techniques (e.g. neural networks (Geisler et al. 2001)). Our approach shares the same advantages as any ILP approach versus a non logic-based machine learning technique: learned hypotheses are structured, human readable and can express relational concepts such as minimising the instances of particular combinations of predicates. Existing background knowledge can be taken into account to capture predefining concepts and the search can be steered towards particular types of hypotheses using a language bias. Furthermore, ILASP2 is also capable of learning preferences with weights and priorities, meaning that more structured and complex preferences can be learned.

An example of the use of an ILP system for learning constraints has been recently presented in (Lallouet et al. 2010) where timetabling constraints are learned from positive and negative examples. In this case the learned rules are hard constraints (e.g., enforcing that a teacher is not in two places at once). Examples of this kind are already computable by ILP_{LAS}, and so are also computable by ILP_{LOAS}.

7 Conclusion and Future Work

We have presented a new framework for ILP, Learning from Ordered Answer Sets, which extends previous ILP systems in that it is able to learn weak constraints and can be used to perform preference learning. The framework can represent partial examples under a brave and a cautious semantics. We have also put forward a new algorithm, ILASP2, that can solve any ILP_{LOAS} task for optimal solutions. This algorithm extends previous work for solving the simpler task ILP_{LAS} and resolves some of the scalability issues associated with the previous algorithm. Some scalability issues remain, especially when there is a particularly large hypothesis space and future work will focus on overcoming these.

References

Banbara, M., Soh, T., Tamura, N., Inoue, K., and Schaub, T. 2013. Answer set programming as a modeling language for course timetabling. *Theory and Practice of Logic Programming 13,* 4-5, 783–798.

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

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., and Schaub, T. 2013. Asp-core-2 input language format.

Corapi, D., Russo, A., and Lupu, E. 2010. Inductive logic programming as abductive search. In *ICLP (Technical Communications)*. 54–63.

Corapi, D., Russo, A., and Lupu, E. 2012. Inductive logic programming in answer set programming. In *Inductive Logic Programming*. Springer, 91–97.

Dastani, M., Jacobs, N., Jonker, C. M., and Treur, J. 2001. Modeling user preferences and mediating agents in electronic commerce. In *Agent Mediated Electronic Commerce*. Springer, 163–193.

Denecker, M., Vennekens, J., Bond, S., Gebser, M., and Truszczyński, M. 2009. The second answer set programming competition. In *Logic Programming and Nonmonotonic Reasoning*. Springer, 637–654.

Fürnkranz, J. and Hüllermeier, E. 2003. Pairwise preference learning and ranking. In *Machine Learning: ECML 2003*. Springer, 145–156.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. 2011. Potassco: The Potsdam answer set solving collection. *AI Communications 24,* 2, 107–124.

Geisler, B., Ha, V., and Haddawy, P. 2001. Modeling user preferences via theory refinement. In *Proceedings of the 6th international conference on Intelligent user interfaces*. ACM, 87–90.

Horváth, T. 2012. A model of user preference learning for content-based recommender systems. *Computing and informatics 28*, 4, 453–481.

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

Lallouet, A., Lopez, M., Martin, L., and Vrain, C. 2010. On learning constraint problems. In *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*. Vol. 1. IEEE, 45–52.

Law, M., Russo, A., and Broda, K. 2014. Inductive learning of answer set programs. In *Logics in Artificial Intelligence (JELIA 2014)*. Springer.

Law, M., Russo, A., and Broda, K. 2015a. Proof of the soundness and completeness of ilasp2. `https://www.doc.ic.ac.uk/~ml1909/Proofs_for_ILASP2.pdf`.

Law, M., Russo, A., and Broda, K. 2015b. Simplified reduct for choice rules in asp. Tech. Rep. DTR2015-2, Imperial College of Science, Technology and Medicine, Department of Computing.

Muggleton, S. 1991. Inductive logic programming. *New generation computing 8,* 4, 295–318.

Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., and Srinivasan, A. 2012. Ilp turns 20. *Machine Learning 86,* 1, 3–23.

Muggleton, S. and Lin, D. 2013. 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, 1551–1557.

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

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

Ray, O., Broda, K., and Russo, A. 2004. A hybrid abductive inductive proof procedure. *Logic Journal of IGPL 12,* 5, 371–397.

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

Srinivasan, A. 2001. The aleph manual. *Machine Learning at the Computing Laboratory, Oxford University*.

Appendix A The ILASP2 Meta Encoding

We present here the ILASP2 meta encoding which is omitted from the main paper. We first summarise some notation used in the encoding.

We will write $body^{+}(R)$ and $body^{-}(R)$ to refer to the positive and negative (respectively) literals in the body of a rule R. Given a program P, $weak(P)$ denotes the weak constraints in P and $non\_weak(P)$ denotes the set of rules in P which are not weak constraints.

Definition 9
For any ASP program P, predicate name *pred* and term $\mathtt{term}$ we will write $reify(P, pred, term)$ to mean the program constructed by replacing every atom $\mathtt{a} \in P$ by $\mathtt{pred(a, term)}$. We will use the same notation for sets of literals/partial interpretations, so for a set S: $reify(S, pred, term) = \{\mathtt{pred(atom, term)} : \mathtt{atom} \in S\}$.

Definition 10
For any ASP program P and any atom $\mathtt{a}$, $append(P, a)$ is the program constructed by appending $\mathtt{a}$ to every rule in P.

Definition 11
Given any term t and any positive example e, $cover(e, t)$ is the program:

$$\mathtt{cov(t)\text{:-}in\_as(e_1^{inc},t)}, \ldots, \mathtt{in\_as(e_n^{inc},t), not\ in\_as(e_1^{exc},t)}, \ldots, \mathtt{not\ in\_as(e_m^{exc},t).}$$
$$\mathtt{\text{:-}not\ cov(t).}$$

Definition 12
Let p_1 and p_2 be distinct predicate names. Given R, a weak constraint $:\sim \mathtt{b_1}, \ldots, \mathtt{b_m}$, not $\mathtt{c_1}, \ldots,$ not $\mathtt{c_l}.[\mathtt{wt@lev}, \mathtt{t_1}, \ldots, \mathtt{t_n}]$, $meta_{weak}(R, p_1, p_2)$ is the rule:

$$\mathtt{w(wt, lev, args(t_1, \ldots, t_n), X)\text{:-}p_2(t), p_1(b_1, X)}, \ldots, \mathtt{p_1(b_m, X), not\ p_1(c_1, X)}, \ldots, \mathtt{not\ p_1(c_l, X).}$$

For a set of weak constraints W, $meta_{weak}(W, p_1, p_2)$ is the set $\{meta_{weak}(R, p_1, p_2) \mid R \in W\}$.

Now that we have defined the predicate w to represent $weak(P, A)$ for each answer set A, we can use some additional rules to determine, given two interpretations, whether one dominates another.

Definition 13

Given any two terms $t1$ and $t2$, $dominates(t1, t2)$ is the program:

$$\left\{\begin{array}{l}
\texttt{dom\_lv(t1,t2,L):-lv(L),\#sum\{w(W,L,A,t1) = W,w(W,L,A,t2) = -W\} < 0.}\\
\texttt{non\_dom\_lv(t1,t2,L):-lv(L),\#sum\{w(W,L,A,t2) = W,w(W,L,A,t1) = -W\} < 0.}\\
\texttt{non\_bef(t1,t2,L):-lv(L),lv(L2),L < L2,non\_dom\_lv(t1,t2,L2).}\\
\texttt{dom(t1,t2):-dom\_lv(t1,t2,L),not non\_bef(t1,t2,L).}
\end{array}\right\}$$

The intuition is that $dom(as1, as2)$ (where $as1$ and $as2$ represent two answer sets A_1 and A_2) should be true if and only if A_1 dominates A_2.

A.1 Encoding the search for positive hypotheses: T_{meta}

Definition 14

Let T be the ILP_{LOAS} task $\langle B, S_M, E^+, E^-, O^b, O^c\rangle$. Then $T_{meta} = meta(B) \cup meta(S_M) \cup meta(E^+) \cup meta(E^-) \cup meta(O^b) \cup meta(O^c)$ where each meta component is as follows:

- $meta(B) = append(reify(non\_weak(B), in\_as, X), as(X)) \cup meta_{weak}(weak(B), in\_as, as, X)$.
- $meta(S_M) = \{append(append(reify(R, in\_as, X), as(X)), in\_h(R_{id})) \mid R \in non\_weak(S_M)\}$
  $\cup \{append(W, in\_h(W_{id})) \mid W \in meta_{weak}(weak(S_M), in\_as, as, X)\}$
  $\cup \{\texttt{:}\sim \texttt{in\_h(R}_{\texttt{id}}\texttt{).[}2 * |\texttt{R}|\texttt{@0,R}_{\texttt{id}}\texttt{]} \mid R \in S_M\}$
  $\cup \{\ \{\texttt{in\_h(R}_{\texttt{id}}\texttt{) : R} \in \texttt{S}_{\texttt{M}}\}\texttt{.}\ \}$
- $meta(E^+) = \left\{\begin{array}{l} cover(e, e_{id}) \\ \texttt{as(e}_{\texttt{id}}\texttt{).}\end{array}\ \middle|\ \langle e^{inc}, e^{exc}\rangle \in E^+\right\}$
- $meta(E^-) = \left\{\begin{array}{l} \texttt{v\_i:-in\_as(e}^{\texttt{inc}}_{\texttt{1}}\texttt{,n),}\ldots\texttt{,in\_as(e}^{\texttt{inc}}_{\texttt{n}}\texttt{,n),}\\ \quad\texttt{not in\_as(e}^{\texttt{exc}}_{\texttt{1}}\texttt{,n),}\ldots\texttt{,}\\ \quad\texttt{not in\_as(e}^{\texttt{exc}}_{\texttt{m}}\texttt{,n).}\\ \texttt{as(n).}\end{array}\ \middle|\ \langle e^{inc}, e^{exc}\rangle \in E^-\right\}$
  $\cup \left\{\begin{array}{l}\texttt{violating:-v\_i.}\\ \texttt{:}\sim\texttt{ not violating.[1@0]}\end{array}\right\}$
- $meta(O^b) = \left\{\begin{array}{l} \texttt{as(o}_{\texttt{id1}}\texttt{).}\quad \texttt{as(o}_{\texttt{id2}}\texttt{).}\\ cover(e^1, o_{id1})\\ cover(e^2, o_{id2})\\ dominates(o_{id1}, o_{id2})\\ \texttt{:-not dom(o}_{\texttt{id1}}\texttt{,o}_{\texttt{id2}}\texttt{).}\end{array}\ \middle|\ o = \langle e_1, e_2\rangle \in O^b\right\} \cup \{\texttt{lv(l).} \mid l \in L\}$
- $meta(O^c) = \left\{\begin{array}{l} dominates(e_1, e_2)\\ \texttt{v\_p(e}^{\texttt{1}}_{\texttt{id}}\texttt{,e}^{\texttt{2}}_{\texttt{id}}\texttt{):-}\\ \quad\texttt{not dom(e}_{\texttt{1}}\texttt{,e}_{\texttt{2}}\texttt{).}\end{array}\ \middle|\ \langle e_1, e_2\rangle \in O^c\right\} \cup \{\texttt{v\_p:-v\_p(T1,T2).}\}$

Example 5

$$B = \left\{\begin{array}{l}
\texttt{p(V):-r(V),not q(V).}\\
\texttt{q(V):-r(V),not p(V).}\\
\texttt{r(1).}\qquad\texttt{r(2).}\\
\texttt{a:-not b.}\\
\texttt{b:-not a.}
\end{array}\right\}
\qquad
S_M = \left\{\begin{array}{l}
\texttt{q(1).}\\
\texttt{:}\sim\texttt{ q(V).[1@1,V,r2]}\\
\texttt{:}\sim\texttt{ b.[1@1,b,r3]}
\end{array}\right\}$$

$$E^{+} = \left\{ \begin{array}{l} \langle\{\mathtt{p(2)}\},\emptyset\rangle, \\ \langle\emptyset,\{\mathtt{p(2)}\}\rangle, \\ \langle\{\mathtt{a}\},\{\mathtt{b}\}\rangle, \\ \langle\emptyset,\{\mathtt{a}\}\rangle \end{array} \right\}$$

$$E^{-} = \left\{ \ \langle\{\mathtt{p(1)}\},\emptyset\rangle \ \right\}$$
$$O^{b} = \left\{ \ \langle e^{+}_{3}, e^{+}_{4}\rangle \ \right\}$$
$$O^{c} = \left\{ \ \langle e^{+}_{1}, e^{+}_{2}\rangle \ \right\}$$

Figure A 1 shows T_{meta}.

```
% meta(B)
in_as(p(V),X) :- in_as(r(V),X),
  not in_as(q(V),X), as(X).
in_as(q(V),X) :- in_as(r(V),X),
  not in_as(p(V),X), as(X).
in_as(r(1),X) :- as(X).
in_as(r(2),X) :- as(X).
in_as(a,X) :- not in_as(b,X), as(X).
in_as(b,X) :- not in_as(a,X), as(X).

% meta(S_M)
in_as(q(1),X) :- as(X), in_h(r1).
w(1,1,args(V,r2),X) :- in_as(q(V),X),
  as(X), in_h(r2).
w(1,1,args(b,r3),X) :- in_as(b,X),
  as(X), in_h(r3).
0 {in_h(r1), in_h(r2), in_h(r3)} 2.
:~ in_h(r1).[2@0,r1]
:~ in_h(r2).[2@0,r2]
:~ in_h(r3).[2@0,r3]

% meta(E^+)
as(1).
as(2).
as(3).
as(4).
cov(1) :- in_as(p(2),1).
cov(2) :- not in_as(p(2),2).
cov(3) :- in_as(a,3), not in_as(b,3).
cov(4) :- not in_as(a,4).
:- not cov(1).
:- not cov(2).
:- not cov(3).
:- not cov(4).

% meta(E^-)
v_i :- in_as(p(1),n).
as(n).
violating :- v_i.
:~ not violating.[1@0, violating]

% meta(O^b)
as(5).
as(6).
cov(5) :- in_as(a,5), not in_as(b,5).
cov(6) :- not in_as(a,6).
:- not cov(5).
:- not cov(6).
dom_lv(5,6,L) :- lv(L),
  #sum{w(W,L,A,5)=W, w(W,L,A,6)=-W} < 0.
wrong_dom_lv(5,6,L) :- lv(L),
  #sum{w(W,L,A,6)=W, w(W,L,A,5)=-W} < 0.
wrong_bef(5,6,L) :- lv(L), L < L2,
  wrong_dom_lv(5,6,L).
dom(5,6) :- dom_lv(5,6,L),
  not wrong_bef(5,6,L).
:- not dom(5,6).
lv(1).

% meta(O^c)
dom_lv(1,2,L) :- lv(L),
  #sum{w(W,L,A,1)=W, w(W,L,A,2)=-W} < 0.
wrong_dom_lv(1,2,L) :- lv(L),
  #sum{w(W,L,A,2)=W, w(W,L,A,1)=-W} < 0.
wrong_bef(1,2,L) :- lv(L), L < L2,
  wrong_dom_lv(1,2,L).
dom(1,2) :- dom_lv(1,2,L),
  not wrong_bef(1,2,L).
v_p(1,2) :- not dom(1,2).
v_p :- v_p(X,Y).
violating :- v_p.
```

Fig. A 1: An example of T_{meta}.

A.2 Encoding classes of violating hypotheses: VR_{meta}

Definition 15
Given any choice rule $R = \texttt{l}\{\texttt{h}_1,\ldots,\texttt{h}_n\}\texttt{u:-body}$, *reductify*$(R)$ is the program:

$$\left\{\begin{array}{c}
\texttt{mmr(h}_1\texttt{,X):-reify(body}^+\texttt{,mmr,X),reify(body}^-\texttt{,not in\_vs,X),l}\{\texttt{h}_1,\ldots,\texttt{h}_n\}\texttt{u,in\_vs(h}_1\texttt{,X)}\cdot\\
\ldots\\
\texttt{mmr(h}_n\texttt{,X):-reify(body}^+\texttt{,mmr,X),reify(body}^-\texttt{,not in\_vs,X),l}\{\texttt{h}_1,\ldots,\texttt{h}_n\}\texttt{u,in\_vs(h}_n\texttt{,X)}\cdot\\
\texttt{mmr(}\bot\texttt{,X):-reify(body}^+\texttt{,mmr,X),reify(body}^-\texttt{,not in\_vs,X),u}+\texttt{1}\{\texttt{h}_1,\ldots,\texttt{h}_n\}\cdot\\
\texttt{mmr(}\bot\texttt{,X):-reify(body}^+\texttt{,mmr,X),reify(body}^-\texttt{,not in\_vs,X),}\{\texttt{h}_1,\ldots,\texttt{h}_n\}\texttt{l}-\texttt{1}\cdot
\end{array}\right\}$$

Definition 16
Let P be an ASP program such that P_1 is the set of normal rules in P, P_2 is the set of constraints in P and P_3 is the set of choice rules in P.

$$\begin{array}{l}
\textit{reductify}(P) = \left\{\begin{array}{l}\texttt{mmr(head(R),X):-reify(body}^+\texttt{(R),mmr,X),}\\ \quad\texttt{reify(body}^-\texttt{(R),not in\_vs,X),vs(X).}\end{array}\middle| R \in P_1\right\}\\
\quad\cup\left\{\begin{array}{l}\texttt{mmr(}\bot\texttt{,X):-reify(body}^+\texttt{(R),mmr,X),}\\ \quad\texttt{reify(body}^-\texttt{(R),not in\_vs,X),vs(X).}\end{array}\middle| R \in P_2\right\}\\
\quad\cup\{\textit{reductify}(R) \mid R \in P_3\}\cdot
\end{array}$$

Definition 17
Let T be the ILP_{LOAS} task $\langle B, S_M, E^+, E^-, O^b, O^c\rangle$ and VR be the set of violating reasons $VI \cup VP$, where VI are violating interpretations and VP are violating pairs.

$VR_{meta}(T)$ is the program $meta(VI) \cup meta(VP) \cup meta(Aux)$ where the meta components are defined as follows:

- $meta(VI) = \left\{\begin{array}{l}\textit{reify}(I, \textit{in\_vs}, I_{id})\\ \texttt{:-not nas(I}_{\texttt{id}}\texttt{).}\\ \texttt{vs(I}_{\texttt{id}}\texttt{).}\end{array}\middle| I \in VI\right\}$
- $meta(VP) = \left\{\begin{array}{l}\textit{dominates}(vp_{id1}, vp_{id2})\\ \textit{reify}(I_1, \textit{in\_vs}, vp_{id1})\\ \textit{reify}(I_2, \textit{in\_vs}, vp_{id2})\\ \texttt{vs(vp}_{\texttt{id1}}\texttt{).}\\ \texttt{vs(vp}_{\texttt{id2}}\texttt{).}\\ \texttt{:-not nas(vp}_{\texttt{id1}}\texttt{),not nas(vp}_{\texttt{id2}}\texttt{),}\\ \quad\texttt{not dom(vp}_{\texttt{id1}}\texttt{,vp}_{\texttt{id2}}\texttt{).}\end{array}\middle| vp = \langle I_1, I_2\rangle \in VP\right\}$
- $meta(Aux) =$

$$\left\{\begin{array}{l}
\textit{reductify}(B)\\
\quad\cup\left\{\begin{array}{l}\texttt{nas(X):-in\_vs(ATOM,X),not mmr(ATOM,X).}\\ \texttt{nas(X):-not in\_vs(ATOM,X),mmr(ATOM,X).}\end{array}\right\}\\
\quad\cup\{\textit{append}(\textit{reductify}(R), \textit{in\_hyp}(R_{id})) \mid R \in \textit{non\_weak}(S_M)\}\\
\quad\cup\{\textit{append}(\textit{meta}_{weak}(W, \textit{in\_vs}, \textit{vs}, X), \textit{in\_hyp}(W_{id})) \mid W \in \textit{weak}(S_M)\}\\
\quad\cup\{\texttt{lv(l).} \mid l \in L\}
\end{array}\right\}$$

Example 6
Recall B, S_M, E^+, E^-, O^b and O^c from example 5 and let VI be the set containing the violating interpretation $\{\texttt{p(1),p(2),r(1),r(2),a}\}$, VP the set containing the violating pair $\langle\{\texttt{p(2),q(1),r(1),r(2),a}\},\{\texttt{q(1),q(2),r(1),r(2),a}\}\rangle$ and let VR be the set of violating reasons $VI \cup VP$. Then figure A 2 shows $VR_{meta}(T)$.

```
% meta(VI)
in_vs(p(1),v1).
in_vs(p(2),v1).
in_vs(r(1),v1).
in_vs(r(2),v1).
in_vs(a,v1).
vs(v1).
:- not nas(v1).

% meta(VP)
in_vs(p(2),v2).
in_vs(q(1),v2).
in_vs(r(1),v2).
in_vs(r(2),v2).
in_vs(a,v2).
vs(v2).

in_vs(q(1),v3).
in_vs(q(2),v3).
in_vs(r(1),v3).
in_vs(r(2),v3).
in_vs(a,v3).
vs(v3).

dom_lv(v2,v3,L) :- lv(L),
   #sum{w(W,L,A,v2)=W,
        w(W,L,A,v3)=-W} < 0.
wrong_dom_lv(v2,v3,L) :- lv(L),
   #sum{w(W,L,A,v3)=W,
        w(W,L,A,v2)=-W} < 0.
wrong_bef(v2,v3,L) :- lv(L), L < L2,
   wrong_dom_lv(1,2,L).
dom(v2,v3) :- dom_lv(v2,v3,L),
   not wrong_bef(v2,v3,L).

:- not nas(v2), not nas(v3),
   not dom(v2,v3).

% meta(Aux)
mmr(p(V),X) :- mmr(r(V),X),
  not in_vs(q(V),X), vs(X).
mmr(q(V),X) :- mmr(r(V),X),
  not in_vs(p(V),X), vs(X).
mmr(r(1),X) :- vs(X).
mmr(r(2),X) :- vs(X).
mmr(a,X) :- not in_vs(b,X), vs(X).
mmr(b,X) :- not in_vs(a,X), vs(X).

mmr(q(1),X) :- vs(X), in_h(r1).
w(1,1,ts(V),X) :- vs(X),
  in_vs(q(V),X), in_h(r2).
w(1,1,args(b,r3),X) :- vs(X),
  in_vs(b,X), in_h(r3).

nas(X) :- in_vs(A,X), not mmr(A,X).
nas(X) :- not in_vs(A,X), mmr(A,X).
```

Fig. A 2: An example of $VR_{meta}(T)$.