Inductive Learning of Answer Set Programs

Mark Law, Alessandra Russo*, and Krysia Broda

Department of Computing, Imperial College London, United Kingdom
{mark.law09,a.russo,k.broda}@imperial.ac.uk

Abstract. Existing work on Inductive Logic Programming (ILP) has focused mainly on the learning of definite programs or normal logic programs. In this paper, we aim to push the computational boundary to a wider class of programs: Answer Set Programs. We propose a new paradigm for ILP that integrates existing notions of brave and cautious semantics within a unifying learning framework whose inductive solutions are Answer Set Programs and examples are partial interpretations We present an algorithm that is sound and complete with respect to our new notion of inductive solutions. We demonstrate its applicability by discussing a prototype implementation, called ILASP (Inductive Learning of Answer Set Programs), and evaluate its use in the context of planning. In particular, we show how ILASP can be used to learn agent's knowledge about the environment. Solutions of the learned ASP program provide plans for the agent to travel through the given environment.

Keywords: Inductive Reasoning, Learning Answer Set Programs, Non-monotonic Inductive Logic Programming.

1 Introduction

For more than two decades, Inductive Logic Programming (ILP) [10] has been an area of much interest. Significant advances have been made both on new algorithms and systems (e.g. [15,8,1,11,12]) and proposals of new logical frameworks for inductive learning (e.g. [13,16]). In most of these approaches an inductive learning task is defined as the search for an hypothesis that, together with a given background knowledge, explains a set of observations (i.e. examples). Observations are usually grouped into positive (E^+) and negative (E^-) examples, and an *inductive solution* is defined as an hypothesis H that is consistent with the background knowledge B and that, together with B, entails the positive examples ($B \cup H \models e$ for every $e \in E^+$) and does not entail the negative examples.

As stated in [16], this semantic view of inductive learning may be too "strong". When $B \cup H$ accepts more than one (minimal) model, it restricts solutions to be only those hypotheses H for which the given observations are true in the intersection of all models of $B \cup H$. Both Brave Induction [16] and Induction of Stable Models [13] applied induction to the stable model semantics [6] such that in situations when $B \cup H$ has more than one stable model, it is just necessary to guarantee that each example is true in at least one stable model of $B \cup H$.

* This research is partially funded by the 7th Framework EU-FET project 600792 "ALLOW Ensembles", and the EPSRC project EP/K033522/1 "Privacy Dynamics".

Their notion of examples is, however, very specific: in [16] there is only one example defined as a conjunction of atoms, and in [13] examples are partial interpretations. When $B \cup H$ has multiple stable models, literals may be true in all models, some of, or none of them, and sometimes only a specified number of a particular set of literals should be true. Neither Brave Induction, nor Induction of Stable Models, is able to express through examples that a literal should be true in all/no stable models. To allow for hypotheses that are ASP programs, a more expressive notion of examples and inductive solution is therefore needed.

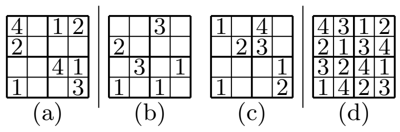

Fig. 1. (a) valid partial grid; (b/c) invalid partial grids; (d) valid complete grid

Consider a simplified version of a sudoku game where the grid includes only sixteen cells. Let us assume that basic definitions of *cell*, *same_row*, *same_col* and *same_block* (true only for two *different* cells in the same row/column/block) are given as background knowledge B expressed as an ASP program, and that the task is to learn an hypothesis H such that the Answer Sets of $B \cup H$ correspond to the valid sudoku grids. A possible hypothesis would be the ASP program:

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

To learn this program, single literal examples such as $value(1, cell(1,1))$ would not be enough, as $value(1, cell(1,1))$ being valid depends on the values of the other cells. Examples should therefore be (partial) grids, e.g. Figure 1(a), and the learned hypothesis, H, should be such that for every example E, $B \cup H$ has an Answer Set corresponding to a complete grid that extends E. It is not sufficient to consider *only* (positive) examples of what *should* be an Answer Set of $B \cup H$: no matter how many examples we give, the hypothesis $0\{value(1, C), value(2, C), value(3, C), value(4, C)\}4 \leftarrow cell(C)$ will always be in the solution space. Each valid sudoku board would be an Answer Set of this hypothesis; however, this is also true for invalid boards, such as those in Figure 1 (b) and (c). What is needed is the use of negative examples. In the sudoku game, negative examples would be invalid partial boards (e.g., Figure 1 (b) and (c)).

In Section 3 we propose a new paradigm for inductive learning, called *Learning from (partial) Answer Sets*. Our approach integrates notions of brave and cautious semantics within a unifying learning framework whose inductive solutions are ASP programs and both positive and negative examples are (partial) interpretations. Inductive solutions are ASP programs that together with a given background knowledge B have at least one Answer Set extending each positive example (this could be a different Answer Set for each example), and no Answer

Set which extends any negative example. The use of negative examples is what differentiates our approach from Brave Induction or Induction of Stable Models. In fact, neither of these two existing approaches would be able to learn the three constraints for the sudoku problem, but our approach can solve any Brave Induction or Induction of Stable Models task. In addition, in our framework negative examples drive the learning of constraints, or the learning of bounds on aggregates. In Section 4 we present our algorithm, *ILASP*, and argue its soundness and completeness with respect to our new notion of inductive learning. In Section 5 we investigate its applicability to a planning problem. We conclude the paper with a review of the related work and a discussion of future directions.

2 Background

We assume the following subset of the ASP language. A *literal* can be either an atom p or its *default negation* not p (often called *negation as failure*). A normal rule is of the form $h \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ where h is called the *head*, $b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ (collectively) is called the *body*, and all h, b_i, and c_j are atoms. A *constraint* is of the form $\leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$. A *choice rule* is an expression of the form $l\{h_1, \ldots, h_m\}u \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ where the head $l\{h_1, \ldots, h_m\}u$ is called an *aggregate*. In an aggregate l and u are integers and h_i, for $1 \leq i \leq m$, are atoms. A variable V occurring in a rule R is said to be *safe* if V occurs in at least one positive literal in the body of R; for example, X is not safe in the rules $p(X) \leftarrow q(Y),\ \text{not } r(Y)$; or $p \leftarrow q,\ \text{not } r(X)$.

An Answer Set Program P is a finite set of normal rules, constraints and choice rules. Given an ASP program P, the Herbrand Base of P, denoted as HB_P, is the set of all ground (variable free) atoms that can be formed from the predicates and constants that appear in P. When P includes only normal rules, a set $A \subseteq HB_P$ is an *Answer Set* of P iff it is the minimal model of the *reduct* P^A (constructed from the grounding of P by removing any rule whose body contains a literal not c_i where $c_i \in A$, and removing any negative literals in the remaining rules). An Answer Set satisfies a ground constraint $\leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ if $\{b_1, \ldots, b_n\} \nsubseteq A$ or $A \cap \{c_1, \ldots c_m\} \neq \emptyset$. Informally, given a ground choice rule $l\{h_1, \ldots, h_m\}u \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ if the body is satisfied by A, then the rule has the effect of generating all Answer Sets in which $l \leq |A \cap \{h_1, \ldots, h_m\}| \leq u$. For a formal definition of the semantics of choice rules, the reader is referred to [5]. Throughout the paper we will denote with $AS(P)$ the set of all Answer Sets of P.

Definition 1. *A partial interpretation E is a pair $E = \langle E^{inc}, E^{exc} \rangle$ of sets of ground atoms, called the* inclusions *and* exclusions *respectively. An Answer Set A extends $\langle E^{inc}, E^{exc} \rangle$ if and only if $(E^{inc} \subseteq A) \wedge (E^{exc} \cap A = \emptyset)$.*

A partial interpretation E is *bravely* entailed by a program P if and only if there exists an Answer Set $A \in AS(P)$ such that A extends E. E is *cautiously* entailed by P if and only if every Answer Set $A \in AS(P)$ extends E.

3 Learning from Answer Sets

In this section we formalize our new paradigm of *Learning from (partial) Answer Sets*. We assume background knowledge and hypotheses to be ASP programs expressed using the ASP language defined in Section 2.

In an ILP task, the expressivity of the hypothesis space is defined by the *language bias* of the task, often characterised by mode declarations [11]. A language 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 called the *head* (resp. *body*) *mode declarations*. Each mode declaration $m_h \in M_h$ (resp. $m_b \in M_b$) is a literal whose abstracted arguments are either v or c. Informally, an atom is said to be *compatible* with a mode declaration m if each instance of v in m is replaced by a variable, and every c by a constant. Given a language bias $M = \langle M_h, M_b \rangle$, a rule of the form $h \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots not\ c_m$ is in the search space S_M if and only if (i) h is empty; or h is an atom compatible with a mode declaration in M_h; or h is an aggregate $l\{h_1, ...h_k\}u$ such that $0 \leq l \leq u \leq k$ and $\forall i \in [1,k]$ h_i are compatible with mode declarations 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. Each rule R in S_M is given a unique identifier R_{id}.

Example 1. Let M be the mode declarations $\langle\{value(c,v)\}, \{cell(v), value(v,v), same\_block(v,v), same\_row(v,v), same\_col(v,v)\}\rangle$. Then the following are in S_M: $value(1,C) \leftarrow cell(C); 1\{value(1,C), value(2,C)\}2 \leftarrow cell(C); \leftarrow value(X,C1), value(X,C2), same\_block(C1,C2)$; whereas the following are not: $value(C) \leftarrow cell(C); cell(C) \leftarrow cell(C); \leftarrow value(1,C1), value(1,C2), same\_block(C1,C2)$.[1]

Definition 2. *A* Learning from Answer Sets *task is a tuple* $T = \langle B, S_M, E^+, E^- \rangle$ *where* B *is the background knowledge,* S_M *is the search space defined by a language bias* M*,* E^+ *and* E^- *are sets of partial interpretations called, respectively, the positive and negative examples. An hypothesis* H *is an* inductive solution *of* T *(written* $H \in ILP_{LAS}(T)$*) if and only if:*

1. $H \subseteq S_M$
2. $\forall e^+ \in E^+\ \exists A \in AS(B \cup H)$ *such that* A *extends* e^+
3. $\forall e^- \in E^-\ \nexists A \in AS(B \cup H)$ *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. Let $B = \{p \leftarrow r\}$, $M = \langle\{q,r\},\{p,r\}\rangle$, $E^+ = \{\langle\{p\},\{q\}\rangle, \langle\{q\},\{p\}\rangle\}$ and $E^- = \{\langle\emptyset,\{p,q\}\rangle, \langle\{p,q\},\emptyset\rangle\}$. An inductive solution is the ASP program H given by $H = \{q \leftarrow \text{not } r; r \leftarrow \text{not } q\}$. The Answer Sets of $B \cup H$ are $\{p,r\}$ and $\{q\}$. The former extends the first positive example, the latter extends the second positive example and clearly neither of them extend any negative examples.

[1] Here, and in the rest of the paper, we use ; as a delimiter in sets of rules.

The following example shows that our learning setting, with the search space as defined in example 1, can learn the sudoku problem described in the introduction.

Example 3. Consider again the sudoku problem, with S_M as described in example 1, B containing the definitions of *same_row*, *same_col*, *same_block* and *cell*. Let the examples be as follows:

$$E^+ = \begin{cases} \langle\{value(cell(1,1),1), \\ \quad value(cell(1,2),2), \\ \quad value(cell(1,3),3), \\ \quad value(cell(1,4),4), \\ \quad value(cell(2,3),2) \\ \quad \}, \emptyset\rangle \end{cases} \qquad E^- = \begin{cases} \langle\{value(cell(1,1),1), \\ \quad value(cell(1,3),1)\}, \emptyset\rangle \\ \langle\{value(cell(1,1),1), \\ \quad value(cell(3,1),1)\}, \emptyset\rangle \\ \langle\{value(cell(1,1),1), \\ \quad value(cell(2,2),1)\}, \emptyset\rangle \end{cases}$$

Let $l\{value(C,1), value(C,2), value(C,3), value(C,4)\}u$ be denoted by $agg(l,u)$. The hypothesis $H_1 = \{agg(1,1) \leftarrow cell(C)\}$ is not an inductive solution, whereas $H_2 = \{agg(1,1) \leftarrow cell(C); \leftarrow value(V,C1), value(V,C2), same\_col(C1,C2); \leftarrow value(V,C1), value(V,C2), same\_row(C1,C2); \leftarrow value(V,C1), value(V,C2), same\_block(C1,C2)\}$ is an inductive solution. This shows that our learning task can incentivise the learning of constraints. With the examples as they are, if we take H_3 to be constructed from H_2 by replacing $agg(1,1)$ with $agg(0,1)$, then H_3 is still an inductive solution. Adding the negative example $\langle\emptyset, \{value(cell(1,1),1), value(cell(1,1),2), value(cell(1,1),3), value(cell(1,1),4), \}\rangle$, this is no longer the case. H_2 is an inductive solution, whereas H_3 is not. This shows that ILP_{LAS} is able to incentivise learning bounds on aggregates.

It is common practice in ILP to search for "optimal" hypotheses. This is usually defined in terms of the number of literals in the hypothesis. This does not apply well to hypotheses that include aggregates: the length of $1\{p,q\}1$ (exactly one of p and q is true) would be the same as the length of $0\{p,q\}2$ (none, either or both of p and q is true), but clearly they do not represent similar concepts. To calculate the length of an aggregate we convert it to disjunctive normal form, as this takes into account both the number of Answer Sets that the aggregate generates and the number of literals it uses. For example, $0\{p,q\}2$ is considered as $(p \wedge q) \vee (p \wedge \text{not } q) \vee (\text{not } p \wedge q) \vee (\text{not } p \wedge \text{not } q)$, which has length 8, whereas $1\{p,q\}1$ is considered as $(p \wedge \text{not } q) \vee (\text{not } p \wedge q)$, which has length 4.

Definition 3. *Given an hypothesis H, the* length *of the hypothesis, $|H|$, is the number of literals that appear in H^D, where H^D is constructed from H by converting all aggregates in H to disjunctive normal form.*

Given an ILP_{LAS} learning task $T = \langle B, S_M, E^+, E^-\rangle$, we denote with $ILP^*_{LAS}(T)$ the set of all optimal inductive solutions of T, where optimality is defined in terms of the length of the hypotheses. We will also denote with $ILP^n_{LAS}(T)$ the set of all inductive solutions of T which have length n.

4 Algorithm

In this section we describe our algorithm ILASP (Inductive Learning of Answer Set Programs) and state its soundness and completeness results. Due to space limitation, proofs have been omitted from the paper but they are available in [9].

Our algorithm works by encoding our ILP_{LAS} task into an ASP program. It makes use of two main concepts: *positive solutions* and *violating solutions*. Positive solutions are those hypotheses that, added to the background knowledge, have Answer Sets which extend each positive example. But some positive solutions may still cover negative examples; we call these the *violating solutions*. The underlying idea of our algorithm is to compute every violating solution of a given length, and then use these to generate a set of constraints which, when added to our task program, eliminate the violating solutions. Theorem 1 shows that the remaining positive solutions are indeed the inductive solutions of the given ILP_{LAS} task. ILASP uses the ASP solver clingo[4] to compute these solutions.

Definition 4. *Let $T = \langle B, S_M, E^+, E^- \rangle$ be an ILP_{LAS} task. An hypothesis $H \in positive\_solns(T)$ iff $H \subseteq S_M$ and $\forall e^+ \in E^+\ \exists A \in AS(B \cup H)$ such that A extends e^+. A positive solution $H \in violating\_solns(T)$ iff $\exists e^- \in E^-$ $\exists A \in AS(B \cup H)$ such that A extends e^-. We write $positive\_solns^n(T)$ and $violating\_solns^n(T)$ to denote the positive and violating solutions of length n.*

Example 4. Consider the ILP_{LAS} task $T = \langle B, S_M, E^+, E^- \rangle$ where $B = \{q \leftarrow r\}$, $E^+ = \{\langle \{p\}, \emptyset \rangle, \langle \{q\}, \emptyset \rangle\}$, $E^- = \{\langle \{p, q\}, \emptyset \rangle\}$ and S_M is given by the following rules $\{p;\ r;\ p \leftarrow r;\quad p \leftarrow \text{not } r;\ r \leftarrow \text{not } p\}$[2]. The hypotheses $H_1 = \{p;\ r\}$, $H_2 = \{p \leftarrow r;\ r\}$ and $H_3 = \{p \leftarrow \text{not } r;\ r \leftarrow \text{not } p\}$ are among the positive solutions of T. Each of the first two hypotheses (together with the background knowledge) has one Answer Set: $\{p; q; r\}$. This extends the negative example in T, and so both hypotheses are violating solutions of T. Note that the positive solutions which are also violating solutions are not inductive solutions, whereas the third positive solution, which is not a violating solution is an inductive solution of T. This is a general property proven by Theorem 1.

Theorem 1. *Let $T = \langle B, S_M, E^+, E^- \rangle$ be an ILP_{LAS} learning task. Then $ILP_{LAS}(T) = positive\_solns(T) \backslash violating\_solns(T)$.*

One method to find all inductive solutions of an ILP_{LAS} learning task T would be to generate all *positive inductive solutions* of T, add each solution, in turn, to the background knowledge in T and solve the resulting program to check whether it accepts Answer Sets that extend any negative examples, i.e. whether it is a *violating solution* of T. As, in practice, this would be inefficient, we instead generate the violating solutions first and use these to constrain our search for positive solutions. Inspired by the technique in [2], we encode our ILP_{ASP} learning task as an ASP program whose Answer Sets will provide our positive solutions. But, differently from [2], our encoding uses a meta-level approach that

[2] In subsequent examples we will refer to these rules in S_M with their R_{id} a to e.

allows us to reason about multiple Answer Sets of $B \cup H$, as in our notion of positive solution there might be multiple positive examples that may be extended by different Answer Sets of $B \cup H$.

Specifically, our definition of a positive solution H requires that each positive example $e^+ \in E^+$ has an Answer Set of $B \cup H$ that extends it. We use the atom $e(A, e^+_{id})$ to represent that a literal A is in the Answer Set that extends the positive example e^+ (with unique identifier e^+_{id}). For each $e^+ \in E^+$, the ground fact $ex(e^+_{id})$. Each rule R in the background knowledge and in the given hypothesis space S_M, is rewritten in a meta-level form by replacing each atom A that appears in R with the atom $e(A, X)$ and adding $ex(X)$ to the body of the rule. In this way the evaluation of the rules (in $B \cup H$) can explicitly refer to specific Answer Sets that extend a specific positive example and guide the search accordingly. In the case of negative examples, for an hypothesis H to be a violating solution, it is only necessary that $B \cup H$ cover one negative example. We therefore use only the fact $ex(neg)$ to represent any negative example. We use a predicate $active(R_{id})$, added to the body of each rule $R \in S_M$, where R_{id} is a unique identifier for R. Rules not chosen for the hypothesis will have this condition evaluated to false (and the rule will be vacuously satisfied). Formally, given an Answer Set A, the function $meta^{-1}(A) = \{R \in S_M : active(R_{id}) \in A\}$.

Definition 5. *Let $T = \langle B, S_M, E^+, E^- \rangle$ be an ILP_{LAS} learning task and $n \in \mathbb{N}$. Let R_{id} be a unique identifier for each rule $R \in S_M$ and let e^+_{id} be a unique identifier for each positive example $e^+ \in E^+$. The learning task T is represented as the ASP* task program *$T^n_{meta} = meta(B) \cup meta(S_M) \cup meta(E^+) \cup meta(E^-) \cup meta(Aux, n)$ where each of these five "meta" components are as follows:*

1. *$meta(B)$ is generated from B by replacing every atom A with the atom $e(A, X)$, and by adding the condition $ex(X)$ to the body of each rule.*
2. *$meta(S_M)$ is generated from S_M by replacing every atom A with the atom $e(A, X)$, and by adding the two conditions $active(R_{id})$ and $ex(X)$ to the body of the rule R that matches the correct rule identifier R_{id}.*
3. *$meta(E^+)$ includes for each $ex^+ = \langle \{li_1, \dots, li_h\}, \{le_1, \dots, le_k\} \rangle \in E^+$ the rules:* $\begin{cases} ex(ex^+_{id}); \quad \leftarrow not\ covered(ex^+_{id}); \\ covered(e^+_{id}) \leftarrow e(li_1, ex^+_{id}), \dots, e(li_h, ex^+_{id}), \\ \qquad not\ e(le_1, ex^+_{id}), \dots, not\ e(le_k, ex^+_{id}) \end{cases}$
4. *$meta(E^-)$ includes for each $e^- = \langle \{li_1, \dots, li_h\}, \{le_1, \dots, le_k\} \rangle \in E^-$ the rule:* $violating \leftarrow e(li_1, neg), \dots, e(li_h, neg), not\ e(le_1, neg), \dots, not\ e(le_k, neg)$
5. *$meta(Aux, n)$ includes the ground facts $length(R_{id}, |R|)$ for every rule $R \in S_M$ and the rule $n\ \#sum\{active(R) = X : length(R, X)\}n$ to impose that the total length of the (active) hypothesis has to be n.*

Example 5. Recall the task T in Example 4. T^3_{meta} is as follows:

1. $meta(B) = \{e(q, X) \leftarrow e(r, X), ex(X)\}$
2. $meta(S_M) = \{e(p, X) \leftarrow active(a), ex(X); e(r, X) \leftarrow active(b), ex(X);$ $e(p, X) \leftarrow e(r, X), active(c), ex(X);\ e(p, X) \leftarrow \text{not } e(r, X), active(d), ex(X);$ $e(r, X) \leftarrow \text{not } e(p, X), active(e), ex(X)\}$

3. $meta(E^{+}) = \{covered(1) \leftarrow e(p,1); covered(2) \leftarrow e(q,2);$ $\leftarrow \text{not } covered(1); \leftarrow \text{not } covered(2); ex(1); ex(2)\}$
4. $meta(E^{-}) = \{violating \leftarrow e(p,neg), e(q,neg)\}$
5. $meta(Aux,3) = \{length(a,1); length(b,1); length(c,2); length(d,2);$ $length(e,2);\ 3\ \#\text{sum}\{active(R) = X : length(R,X)\}3\}$

Proposition 1. *Let $T = \langle B, S_M, E^{+}, E^{-}\rangle$ be an ILP_{LAS} task and $n \in \mathcal{N}$. Then $H \in positive\_solns^{n}(T)$ if and only if $\exists A \in AS(T^{n}_{meta})$ such that $H = meta^{-1}(A)$.*

But as stated in Theorem 1, to compute our inductive solution we need also to compute the violating solutions. The same ASP encoding described in Definition 5 can be used to generate all the violating solutions. Specifically, given a length n, the ASP program $T^{n}_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(neg)\}$ will have Answer Sets that include $active(R_{id})$ of hypotheses $R \in S_M$ that are violating solutions. This is captured by Proposition 2.

Proposition 2. *Let $T = \langle B, S_M, E^{+}, E^{-}\rangle$ be an ILP_{LAS} task and $n \in \mathcal{N}$. Let P be the ASP program $T^{n}_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(neg)\}$. Then $H \in violating\_solns^{n}(T)$ if and only if $\exists A \in AS(P)$ such that $H = meta^{-1}(A)$.*

The main idea of our learning algorithm, called ILASP, is to compute first all violating solutions of a given ILP_{LAS} learning task T by solving the ASP program $T^{n}_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(neg)\}$. Then to convert these solutions into constraints[3] and again to solve T^{n}_{meta}, augmented this time with these new constraints. The Answer Sets of this second step will provide all the inductive solutions of T. This is formally described in Algorithm 1.

Algorithm 1. ILASP

```
procedure ILASP($T$)
    $solutions = []$
    for $n = 0$; $solutions.empty$; $n{+}{+}$ do
        $vs = AS(T^{n}_{meta} \cup \{\leftarrow \text{ not } violating;\quad ex(neg)\})$
        $ps = AS(T^{n}_{meta} \cup \{constraint(meta^{-1}(V)) : V \in vs\})$
        $solutions = \{meta^{-1}(A) : A \in ps\}$
    end for
    return $solutions$
end procedure
```

We denote with $ILP^{n}_{LAS}(T)$ the set of all inductive solutions of length n. Proposition 3 states that the Answer Sets of the ASP task program augmented with $constraint(H)$, for each violating solution H, correspond exactly to the inductive solutions of length n of the original learning task.

Proposition 3. *Let $T = \langle B, S_M, E^{+}, E^{-}\rangle$ be an ILP_{LAS} task and $n \in \mathcal{N}$. Let $P = T^{n}_{meta} \cup \{constraint(V) : V \in violating\_solns^{n}(T)\}$. Then a hypothesis $H \in ILP^{n}_{LAS}(T)$ if and only if $\exists A \in AS(P)$ such that $H = meta^{-1}(A)$.*

[3] $constraint(\{R_1, \ldots, R_h\})$ denotes the rule $\leftarrow active(R_{id1}), \ldots, active(R_{idh})$, where $R_{id1}, \ldots R_{idh}$ are the unique identifiers of rules $R_1, \ldots, R_h$ in H.

The following theorem states that $ILASP$ is sound and complete with respect to the notion of optimal[4] inductive solutions in $ILP^{*}_{LAS}(T)$. $ILASP(T)$ denotes the set of hypotheses computed by ILASP for a given ILP_{LAS} task T.

Theorem 2. *Let T be any ILP_{LAS} learning task such that there is at least one inductive solution. Then $ILASP(T) = ILP^{*}_{LAS}(T)$.*

Proof. At each step through the for loop (fix n to be any natural number): let H be an hypothesis of length n and $P = T^{n}_{meta} \cup \{\leftarrow \text{not } violating;\ ex(neg)\}$. By Prop. 2, $H \in violating\_solns^{n}(T)$ iff $\exists A \in AS(P)$ st $H = meta^{-1}(A)$.

$\Rightarrow H \in violating\_solns^{n}(T)$ iff $\exists A \in vs$ st $H = meta^{-1}(A)$.

$\Rightarrow violating\_solns^{n}(T) = \{meta^{-1}(A) : A \in vs\}$

$ps = AS(T^{n}_{meta} \cup \{constraint(meta^{-1}(V)) : V \in vs\})$

$\Rightarrow ps = AS(T^{n}_{meta} \cup \{constraint(V) : V \in violating\_solns^{n}(T)\})$

$\Rightarrow H \in ILP^{n}_{LAS}(T)$ iff $\exists A \in ps$ st $H = meta^{-1}(A)$ by Proposition 3.

$\Rightarrow ILP^{n}_{LAS}(T) = \{meta^{-1}(A) : A \in ps\} = solutions$

As $ILASP(T)$ returns $ILP^{n}_{LAS}(T)$ for the first n for which it is non-empty, $ILASP(T) = ILP^{*}_{LAS}(T)$. □

We are currently working to improve ILASP's scalability. In general there could be many violating solutions before the first inductive solution; for example, with the sudoku problem, the first positive solution is H_1 from example 3 which has length 17; however, the first inductive solution is not until H_2 at length 26. There are many thousands of violating solutions between lengths 17 and 26 (many of these are constructed by adding rules to H_1). By restricting the search space so that the only permitted aggregates are those of the form $agg(l, u)$ for some l and u (see example 3), our implementation was able to find the correct solution. In current work we are exploring techniques to make ILASP more efficient and more scalable. One possibility is to keep a set of full Answer Sets which extend negative examples. When we find a violating solution, we add an Answer Set to this set and rule out further solutions which accept this Answer Set. This set is likely to be much smaller than the set of violating solutions.

5 Application to a Planning Problem

In this section we apply our approach to a planning problem where an agent is in a room at a given position and attempts to get to a target position. Figure 2 gives a graphical representation of the room and the legend describes its main features. The challenge in this planning problem is that although the agent has complete knowledge of the grid map, it does not know the meaning of the various cell features. For instance, it knows which cells are locked, but not that to go through a locked cell it must first visit the key to that cell. The agent's goal is to learn the definition of valid move to allow it to reach the target position.

[4] Note that the optimality – hypotheses with shortest length – is guaranteed by the incremental property of our algorithm.

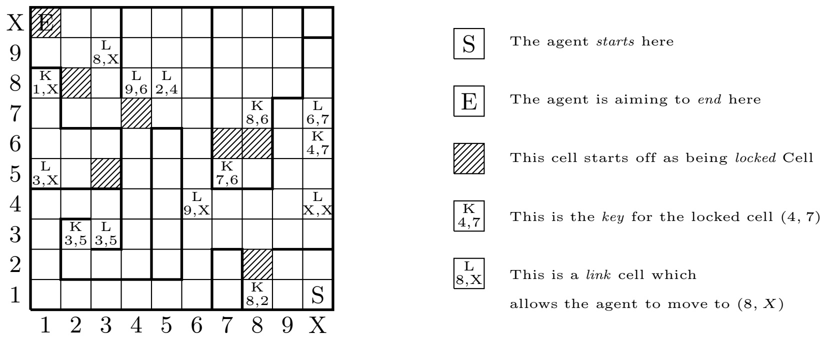
S
The agent *starts* here
E
The agent is aiming to *end* here
This cell starts off as being *locked* Cell
K 4,7
This is the *key* for the locked cell (4, 7)
L 8,X
This is a *link* cell which allows the agent to move to (8, X)

Fig. 2. Cells with diagonal lines are *locked* and the agent must visit the corresponding key before it can enter these cells. *Link* cells allow the agent to jump to the indicated destination cell. The thick black lines represent walls.

The planning problem is modelled as follows. At each step an oracle informs the agent on which cells it could move to next, called the *valid moves.* If the agent, using its current knowledge, infers valid moves that are different from that suggested by the oracle then the agent learns an updated hypothesis; otherwise it plans a path to the target position, using its current hypothesis, and selects as its next move the first move in the plan. By using ASP optimisation, the agent can even plan for the optimal (shortest) plan.[5] In what follows we show three scenarios illustrating three different learning outcomes. In the first scenario, the agent learns just the concept of valid move; in the second scenario, part of the existing background knowledge is removed and the agent has to also learn a new concept that does not appear in the examples or in the background knowledge, showing the ability of ILP_{LAS} to support predicate invention [11]. Finally, in the third scenario, the environment is non deterministic, causing the agent to learn a non deterministic notion of valid move.

Scenario 1: In this simplest scenario, the agent is given the grid map, encoded as facts, together with the history of the cells it has been at from the start and the notions of adjacent cells, visited cell and unlocked cell (given below).

```
unlocked(C, T) :- visited_cell(Key, T), key(C, Key).
unlocked(C, T) :- cell(C), not locked(C), time(T).
```

The task is for the agent to learn the rules:

```
valid_move(C1, T) :- agent_at(C2, T), not wall(C1, C2),
                     adjacent(C1, C2), unlocked(C1, T).
valid_move(C1, T) :- agent_at(C2, T), link(C2, C1), unlocked(C1, T).
```

We denote with VM_{oracle} the set of *valid_move*/2 facts that the oracle generates and with VM_{agent} the set of *valid_move*/2 facts that the agent infers at a given time using its current knowledge and hypothesis. When VM_{oracle} and VM_{agent} differ, the agent uses our *ILASP* algorithm to find a new hypothesis such that

[5] If the agent cannot generate, with its current knowledge, a plan to reach the target position, then optimality is defined in terms of exploration of the map.

VM_{agent} and VM_{oracle} are once again equal. The background knowledge consists of the definitions of *adjacent*, *unlocked*, *visited_cell* and of the history of the cells the agent has been at from the start. In this simple scenario the target program has only one Answer Set, thus only one positive example is necessary. In particular, at each learning step, the positive example is given by every valid move in VM_{oracle} that does not appear in VM_{agent}. These are the moves the agent did not realise were possible, hence it needs to learn. The negative examples are constructed from the moves that are in VM_{agent} but not in VM_{oracle}. These are the moves the agent wrongly thought were possible and that the new learned hypothesis should no longer cover. The first few sets of examples are shown in example 6. Note that the learning task does not take into account the complete history of the valid moves. So it is possible that the new hypothesis wrongly classifies as invalid a move made at an earlier step. If VM_{oracle} is still different to VM_{agent}, the examples are again updated and a new hypothesis is learned.

Example 6. At the first step $VM_{oracle} = \{(9,1),(10,2)\}$, but given the agent's initial hypothesis, $\emptyset$, $VM_{agent} = \emptyset$. The examples are therefore:

$$E^+ = \left\{ \begin{array}{l} \langle\{valid\_move(cell(9,1),1), \\ \quad valid\_move(cell(10,2),1)\},\emptyset\rangle \end{array} \right. \qquad E^- = \emptyset$$

ILASP returns the hypothesis $valid\_move(C,T) \leftarrow unlocked(C,T)$. VM_{agent} and VM_{oracle} are still not equal as VM_{agent} now contains too many moves. The agent therefore extends its examples to:

$$E^+ = \left\{ \begin{array}{l} \langle\{valid\_move(cell(9,1),1), \\ \quad valid\_move(cell(10,2),1) \\ \},\emptyset\rangle \end{array} \right. \quad E^- = \left\{ \begin{array}{l} \langle\{valid\_move(cell(8,1),1)\},\emptyset\rangle, \\ \langle\{valid\_move(cell(7,1),1)\},\emptyset\rangle, \\ \qquad \ldots \end{array} \right.$$

The new hypothesis is $valid\_move(C,T) \leftarrow adjacent(C,C2), agent\_at(C2,T)$. VM_{agent} and VM_{oracle} are now equal, and so the agent makes it's first move (to $(9,1)$). VM_{oracle} and VM_{agent} are equal until the agent reaches $(8,1)$. VM_{agent} now has $(7,1)$ where VM_{oracle} does not. The previous example set is augmented with the new negative example $\langle\{valid\_move(cell(7,1),3)\},\emptyset\rangle$. As shown in Table 1, *ILASP* is able to generate the correct solution in 6 learning steps; however, in this scenario it only had to learn a single predicate. Next we investigate what happens when *ILASP* needs to learn an unseen predicate.

Scenario 2: This scenario differs from the previous one in that the agent is not given the definition of unlocked cell. The language bias of this learning task is therefore augmented with a new predicate, called $extra/2$ added to both M_h and M_b. We expected the agent to learn the previous hypothesis along with the definition of *unlocked*; however, the agent learned the shorter hypothesis:

```
extra(C,T) :- agent_at(C1, V1), link(C1,C).
extra(C,T) :- adjacent(C,C1), agent_at(C1, T), not wall(C,C1).
valid_move(C, T) :- extra(C,T), not locked(C).
valid_move(C, T) :- extra(C,T), key(C1,C), visited_cell(C1,T).
```

So far, due to the deterministic environment the agent has learned programs with only one Answer Set. The next scenario explores a non-deterministic setting.

Table 1. Results for the first scenario. Each row shows the path the agent took while it believed a particular hypothesis ((1..3, 2) abbreviates (1, 2), (2, 2), (3, 2)) .

Path	Hypotheses (With variables renamed for readability)
$(10, 1)$	
$(10, 1)$	$valid\_move(C, T) \leftarrow unlocked(C, T)$.
$(10..8, 1)$	$valid\_move(C, T) \leftarrow adjacent(C, C2), agent\_at(C2, T)$.
$(8, 1..4)$, $(7..6, 4)$	$valid\_move(C, T) \leftarrow \text{not } wall(C, C2), adjacent(C, C2), agent\_at(C2, T)$.
$(6, 4..7)$, $(5, 7)$	$valid\_move(C, T) \leftarrow \text{not } wall(C, C2), adjacent(C, C2), agent\_at(C2, T)$. $valid\_move(C, T) \leftarrow link(C, C2), agent\_at(C2, T)$.
$(5, 7..8)$, $(2..3, 4)$, $(3, 3)$	$valid\_move(C, T) \leftarrow \text{not } wall(C, C2), unlocked(C, T), adjacent(C, C2), agent\_at(C2, T)$. $valid\_move(C, T) \leftarrow link(C, C2), agent\_at(C2, T)$.
$(3, 3)$, $(2..3, 3)$, $(3, 5)$, $(2, 5..6)$, $(1, 6..7)$ $(1, 8..5)$, $(3..1, 10)$	$valid\_move(C, T) \leftarrow \text{not } wall(C, C2), unlocked(C, T), adjacent(C, C2), agent\_at(C2, T)$. $valid\_move(C, T) \leftarrow link(C, C2), unlocked(C, T), agent\_at(C2, T)$.

Scenario 3: We now further complicate matters for our agent by removing the guarantee that the set of valid moves it has is always inferable given its history. The change to the scenario is that *link* is given an extra argument: the *flipped* destination cell (if the destination cell is (X, Y), the flipped cell is (Y, X)). Now whenever the agent lands on a link cell, the oracle decides (randomly) whether to give the destination cell, or the flipped cell as a valid move. In the first two scenarios we restricted the search space to hypotheses without aggregates, whereas here we allow aggregates, which extends the search space to include many more rules. As a consequence, to overcome the scalability issues discussed in the previous section, we needed to make a small addition to the background knowledge that combines the concepts of adjacent and wall into a new concept $joined(C1, C2) \leftarrow adjacent(C1, C2), \text{not } wall$. In this scenario, in addition to the set of valid moves, the oracle also gives to the agent a second set of *potentially* valid moves (the union of all sets of valid moves the oracle could have given).

The fact that the environment is non-deterministic changes the learning task slightly. We can no longer encode every invalid move proposed by the agent at a particular time as a negative example. This is because, had the oracle made a different choice, the move *might* have been a valid one. If an invalid move appears in the set of *potentially valid* moves, then it is instead added to the exclusion set of the positive example at that time. This means that it cannot occur in the Answer Set extending this positive example, but could well appear in other Answer Sets of the program. The agent was able to learn the rules:

```
1 {valid_move(C, T); valid_move(FC, T)} 1 :-
    unlocked(C, T), link(C2, C, FC), agent_at(C2, T).
valid_move(C, T) :- unlocked(C, T), joined(C, C2), agent_at(C2, T).
```

6 Related Work

In this section we review the related work. We reformulate (but preserve the meaning of) some learning tasks to allow for easier comparison with our own.

The goal of traditional ILP has been to learn Prolog style logic programs. Usually this is restricted to learning definite programs (with no negation as failure). The learning task of these traditional ILP systems is equivalent to a Learning from Answer Sets task with a single positive example (and no negative examples) and with the search space restricted to definite logic programs. But learning more general ASP programs rather Prolog programs has the clear advantage that the ASP representation is completely declaratvie. In Prolog, we would have to learn a procedure for constructing valid sudoku boards, whereas in ASP we only learned the rules of sudoku.

Induction of Stable Models [13] extends the definition of ILP to the stable model semantics. An *Induction of Stable Models task* is a tuple $\langle B, S_M, E\rangle$ where B is the background knowledge, S_M is the search space and E is a set of partial interpretations. $H \in ILP_{sm}\langle B, M, E\rangle$ iff (i) $H \subseteq S_M$; and (ii) $\forall O \in E : \exists A \in AS(B \cup H)$ such that A extends O. This is a special case of ILP_{LAS}: with no negative examples. For any B, S_M, E: $ILP_{sm}\langle B, S_M, E\rangle = ILP_{LAS}\langle B, S_M, E, \emptyset\rangle$. However, negative examples are needed to learn Answer Set programs in practice, as otherwise there is no concept of what should not be in an Answer Set. In our planning, for instance, no negative examples would give the optimal solution $0\{valid\_move(C, T)\}1 \leftarrow cell(C), time(T)$ (at any time for each cell C, we may or may not be allowed to move to C). This does cover our positive examples, but it is not specific enough to be useful for planning.

Brave Induction[16] finds an hypothesis which covers a single observation O. A *Brave Induction task*, when defined in the context of ASP, is a tuple $\langle B, S_M, O\rangle$ where B is the background knowledge, S_M the search space and O is a set of atoms. $H \in ILP_b\langle B, M, O\rangle$ iff (i) $H \subseteq S_M$; and (ii) $\exists A \in AS(B \cup H)$ such that $O \subseteq A$. For any B, M, O, the Brave Induction task $ILP_b\langle B, M, O\rangle = ILP_{LAS}\langle B, M, \{\langle O, \emptyset\rangle\}, \emptyset\rangle$. *ASPAL* [2] uses ASP as a solver to compute a solution to a standard ILP Task. ASPAL's learning task, similarly to that of XHAIL [14], is between Brave Induction and Induction of Stable Models. It has a single positive example which is a partial interpretation. ASPAL's method of using an ASP solver to search for the inductive solutions to an ILP task inspired our own. Our method conducts the search in multiple stages however, as we not only require the brave entailment of the positive examples, but also the cautious entailment of the negation of our negative examples. The search spaces in ASPAL and XHAIL did not include aggregates or constraints.

In [3], De Raedt defines *Learning from Partial Interpretations.* Under ILP_{LFPI} an example E (a partial interpretation) is covered by a hypothesis H iff there is a model of $B \cup H$, which extends E. Unlike ILP_{LAS}, as this definition uses models, H covers an example E iff $B \cup H \cup E$ is consistent. Another approach is *Learning from Interpretation Transitions* [7]. The examples here are pairs of interpretations $\langle I, J\rangle$ such that J must equal $T_{B\cup H}(I)$. This task can be mapped to an ILP_{LAS} task by replacing every atom h in the head of rules in B and S_M or occuring in J with the new atom $tp(h)$, representing that $h \in T_{B\cup H}(I)$. Each $I_i \in \{I_1, \ldots, I_n\}$ would then be put in the background knowledge with a condition eg_i in the body (supported by a choice rule $1\{eg_1, \ldots, eg_n\}1$) and eg_i would be added to each J_i which then become the positive examples.

7 Conclusion and Future Work

We have presented a new paradigm for ILP that allows the learning of ASP programs. We have designed and implemented an algorithm which is able to compute inductive solutions, and have shown how it can be used in a planning problem.

There are two avenues of future work: improving the efficiency of our algorithm; and learning a larger subset of the language of ASP. In particular we believe that learning optimisation statements in ASP will facilitate many more applications, as most of ASP's applications involve optimisation.

References

1. Corapi, D., Russo, A., Lupu, E.: Inductive logic programming as abductive search. In: ICLP (Technical Communications), pp. 54–63 (2010)
2. Corapi, D., Russo, A., Lupu, E.: Inductive logic programming in answer set programming. In: Muggleton, S.H., Tamaddoni-Nezhad, A., Lisi, F.A. (eds.) ILP 2011. LNCS, vol. 7207, pp. 91–97. Springer, Heidelberg (2012)
3. De Raedt, L.: Logical settings for concept-learning. Artificial Intelligence 95(1), 187–201 (1997)
4. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 107–124 (2011)
5. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers (2012)
6. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, vol. 88, pp. 1070–1080 (1988)
7. Inoue, K., Ribeiro, T., Sakama, C.: Learning from interpretation transition. Machine Learning 94(1), 51–79 (2014)
8. Kimber, T., Broda, K., Russo, A.: Induction on failure: learning connected horn theories. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 169–181. Springer, Heidelberg (2009)
9. Law, M., Russo, A., Broda, K.: Proofs for inductive learning of answer set programs, `https://www.doc.ic.ac.uk/~ml1909/ILASP_Proofs.pdf`
10. Muggleton, S.: Inductive logic programming. New Generation Computing 8(4), 295–318 (1991)
11. Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., Srinivasan, A.: Ilp turns 20. Machine Learning 86(1), 3–23 (2012)
12. Muggleton, S., Lin, D.: Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 1551–1557. AAAI Press (2013)
13. Otero, R.: Induction of stable models. In: Rouveirol, C., Sebag, M. (eds.) ILP 2001. LNCS (LNAI), vol. 2157, pp. 193–205. Springer, Heidelberg (2001)

14. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic 7(3), 329–340 (2009)
15. Ray, O., Broda, K., Russo, A.: A hybrid abductive inductive proof procedure. Logic Journal of IGPL 12(5), 371–397 (2004)
16. Sakama, C., Inoue, K.: Brave induction: a logical framework for learning from incomplete information. Machine Learning 76(1), 3–35 (2009)