Inductive Learning of Answer Set Programs

Mark Law, Alessandra Russo, and Krysia Broda

Department of Computing, Imperial College London,
{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) Answer Sets. 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 from observation. 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. [16]). In most of these approaches an inductive learning task is defined as the computation of 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 $B \cup H$. Brave Induction [16] partly overcomes this limitation by relaxing the notion of an inductive solution to hypotheses that, together with the background knowledge, accept at least one minimal model satisfying the examples. Both Brave Induction and Induction of Stable Models [13] also 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 $B \cup H$. 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. In particular, this is the case when the hypothesis H is an Answer Set Program (ASP) and the models of $B \cup H$ are Answer Sets. 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 Answer Set Programs, a more expressive notion of examples and inductive solution is therefore needed.

Consider, for instance, a simplified version of a sudoku game where the grid includes only sixteen cells (see Figure 1).

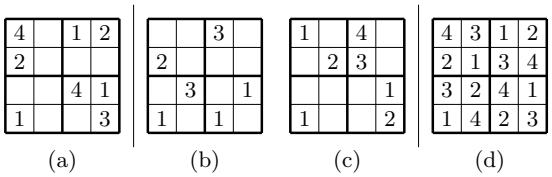

Fig. 1. One valid partial sudoko board (a), followed by two invalid partial boards (b) and (c) and one complete valid board (d)

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 Answer Set Program, and that the task is to learn the set of rules that together with B generates all valid sudoku grids. A possible hypothesis would in this case be the following 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).
```

The first rule states that each cell has exactly one value (1, 2, 3 or 4). The remaining 3 rules are *constraints* stating that no two different cells in the same column/row/block can have the same value. Each Answer Set of this program corresponds to exactly one (complete) valid board.

To learn this program (given the definitions of *cell*, *same_row*, *same_col*, *same_block* as background knowledge) single literal examples, $value(1, cell(1, 1))$, 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) boards, 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 board that extends E. But it is not sufficient to consider *only* (positive) examples of what *should* be included in an Answer Set of the target hypothesis: no matter how many (positive) examples we give the following hypothesis will always be in the solution space:

```
0 { value(1, C), value(2, C), value(3, C), value(4, C) } 4 :- cell(C).
```

Every correct sudoku board would be an Answer Set of the above hypothesis as every correct board has value between 0 and 4 in each cell; however, this is also true for many incorrect boards, such as those in Figure 1 (b) and (c). What is needed is the use of negative examples, also as partial interpretations, and a new notion of inductive learning from *positive* and *negative (partial) interpretations*.

In this paper we propose a new paradigm for inductive learning that allows *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 (or partial Answer Sets). 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 described above as part of an inductive solution for the given sudoku problem, but our approach can solve any Brave Induction or Induction of Stable Models task.

In our framework negative examples drive the learning of constraints, or the learning of bounds on aggregates. In the sudoku game, negative examples would be invalid partial boards (e.g., Figure 1 (b) and (c)). These examples should not be extended by any Answer Set of the learned program. If, for instance, we wanted to force the learning of an hypothesis whose models are only complete boards, we could consider, for instance, a negative example which states that none of the values 1-4 appear in a particular cell. Note that this is different from partial boards (examples) which do not specify a value for a particular cell, as they could still be extended by a board which does.

Section 3 describes our new learning paradigm, called *Learning from (partial) Answer Sets*, by formally defining our new notions of learning task and inductive solutions. 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. In particular we show how ILASP can be used to learn agent's knowledge about its environment from observation. Answer Sets of the learned ASP hypotheses provide in this case plans for the agent to travel through the given environment. We conclude the paper with a review of the related work and a discussion of future directions.

2 Background

In this section we briefly summarize basic notions and terminology that will be used throughout the paper. 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 by 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 the *head* of the rule, $b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ (collectively) is the *body* of the rule, and all h, b_i, and c_j are atoms. A *constraint* is of the form $\leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ where the empty head means *false*. 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.

Definition 1. *A variable V occurring in a rule R is said to be* safe *if V occurs in at least one non defaultly negated literal in the body of R.*

Example 1. In the following rules the variable X is not safe:

$$p(X) \leftarrow q(Y),\ \text{not } r(Y) \qquad\qquad p \leftarrow q,\ \text{not } r(X).$$

An Answer Set Program P is a finite set of normal rules, constraints and choice rules. Given an Answer Set 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, which is the program constructed from the grounding of P by first removing any rule whose body contains a literal not c_i where $c_i \in A$, and then removing any defaultly negated 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 it is not the case that $\{b_1, \ldots, b_n\} \subseteq A$ and $A \cap \{c_1, \ldots c_m\} = \emptyset$. A constraint therefore has the effect of eliminating Answer Sets in which the body of the constraint is true. Finally, informally, if the body of a ground choice rule $l\{h_1, \ldots, h_m\}u \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ is satisfied by A, then the rule has the effect of generating all Answer Sets in which $l \leq |A \cap \{h_1, \ldots, h_m\}| \leq u$. Note that this no longer enforces minimality. For a formal definition of the semantics of Answer Set Programs with choice rules, the reader is referred to [5]. Throughout the paper we will denote with $AS(P)$ the set of all Answer Sets of an Answer Set Program P.

Definition 2. *A partial interpretation E is a pair $E = \langle E^{inc}, E^{exc} \rangle$ of sets of literals, called the* inclusions *and* exclusions *respectively. An Answer Set A extends $\langle E^{inc}, E^{exc} \rangle$ if and only if $(E^{inc} \subseteq A) \wedge (E^{exc} \cap A = \emptyset)$.* [1]

A partial interpretation E is called a *brave* consequence of a program P if and only if there exists an Answer Set $A \in AS(P)$ such that A extends E. E is called a *cautious* consequence of a program 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, and examples to be partial interpretations.

In an ILP task, the expressivity of the hypothesis space is defined by the *language bias* of the task. Mode declarations are a popular means of characterising the language bias [11]. They specify which literals may appear in the head and in the body of a hypothesis. Given a language bias the full hypothesis space, also called *search space* and denoted with S_M, is given by the finite set of all the rules that can be constructed according to the given bias. A language bias

[1] Note: $\langle \{e_1, ..., e_n\}, \{f_1, ..., f_m\} \rangle$ can also be written $\{e_1, ..., e_n, \text{not } f_1, ..., \text{not } f_m\}$.

can be defined as a pair 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 (resp. 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 every instance of v in m has been replaced with a variable, and every c with a constant.

Definition 3. *Given a set of mode declarations $M = \langle M_h, M_b \rangle$, a rule of the form $h \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots \text{not } c_m$ 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 2. Consider again the sudoku example and assume the following mode declarations: $M = \langle \{value(c,v)\}, \{cell(v), value(v,v), same\_block(v,v)\}\rangle$. Then $value(1,C) \leftarrow cell(C) \in S_M$; $1\{value(1,C), value(2,C)\}2 \leftarrow cell(C) \in S_M$; $\leftarrow value(X,C1), value(X,C2), same\_block(C1,C2) \in S_M$. The following rules are not in the search space S_M: $value(C) \leftarrow cell(C)$; $cell(C) \leftarrow cell(C)$; and $\leftarrow value(1,C1), value(1,C2), same\_block(C1,C2)$.

Next we define our new notions of inductive task and inductive solution in the *Learning from Answer Sets* setting.

Definition 4. *A* Learning from Answer Sets *task is a tuple $T = \langle B, S_M, E^+, E^- \rangle$ where B be is the background knowledge, S_M the search space defined by a language bias M, E^+ and E^- are sets of partial interpretations called, respectively, the positive and negative examples. A hypothesis $H \in ILP_{LAS}(T)$, the set of* inductive solutions *of 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 3. Let $B = \{p \leftarrow r\}$, $M = \langle \{q,r\}, \{p,r\}\rangle$, $E^+ = \{\{p, \text{ not } q\}, \{q, \text{ not } p\}\}$ and $E^- = \{\{\text{not } p, \text{not } q\}, \{p,q\}\}$. An inductive solution is the ASP program H given by $H = \{q \leftarrow \text{not } r; r \leftarrow \text{not } q\}$ [2]. 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.

It is common practice in ILP to search for hypotheses which are "most compressed", following the Occam's Razor principle of looking for simplest theories. The notion of compressed is usually defined in terms of the number of literals that appear in the hypothesis. This metric does not apply well to hypotheses that include aggregates. The length of $1\{p,q\}1$ (exactly one of p and q is true)

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

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. So, 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 the disjunction $(p \wedge q) \vee (p \wedge \text{not } q) \vee (\text{not } p \wedge q) \vee (\text{not } p \wedge \text{not } q)$, 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 5. *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 the design and implementation of 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 6. *Let $T = \langle B, S_M, E^+, E^- \rangle$ be an ILP_{LAS} task. An hypothesis $H \in positive\_solutions(T)$, called the set of* positive inductive solutions *of T, if and only if $H \subseteq S_M$ and $\forall e^+ \in E^+ \; \exists A \in AS(B \cup H)$ such that A extends e^+.*

Example 4. Consider the ILP_{LAS} task $T = \langle B, S_M, E^+, E^- \rangle$ where $B = \{q \leftarrow r\}$, $E^+ = \{\{p\}, \{q\}\}$, $E^- = \{\{p, q\}\}$ and S_M is given by the following rules $\{p;\ r;\ p \leftarrow r;\quad p \leftarrow \text{not } r;\ r \leftarrow \text{not } p\}$[3]. 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. Note that only the last solution is an inductive solution of T.

Definition 7. *Let $T = \langle B, S_M, E^+, E^- \rangle$ be an ILP_{LAS} task. An hypothesis $H \in violating\_solutions(T)$, called the set of* violating inductive solutions *of T, if and only if $H \in positive\_solutions(H)$ and $\exists e^- \in E^- \; \exists A \in AS(B \cup H)$ such that A extends e^-.*

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

We will write $positive\_solutions^n(T)$ and $violating\_solutions^n(T)$ to denote the positive and violating inductive solutions of length n.

Example 5. Consider Example 4 again. 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 inductive 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 actually an inductive solution of the given ILP_{LAS} task. 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* $ILP_{LAS}(T) = positive\_solutions(T) \setminus violating\_solutions(T)$.

One method to find all inductive solutions of a 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 the negative example, i.e. *violating solution* of T. But this would in practice be inefficient. Instead, we 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 inductive solutions. But, differently from [2], our encoding uses a meta-level approach that allows us to reason about multiple Answer Sets of $B \cup H$, as in our notion of inductive solution there might be multiple positive examples that may be extended by different Answer Sets of $B \cup H$.

Specifically, our definition of an inductive solution H requires that each positive example $e^+ \in E^+$ has an Answer Set of $B \cup H$ that extends it. We use in our encoding the atom $e(A, e^+_{id})$ to represent that a literal A is in the Answer Set that extends the positive example e^+ (denoted by the unique identifier e^+_{id}). Ground facts of the form $ex(e^+_{id})$ are added to our encoding for each $e^+ \in E^+$ to uniquely identify each positive example $e^+ \in E^+$. 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. This specific reference to examples when computing Answer Sets is only important for the positive examples. In the case of negative examples, for an hypothesis H to be a violating solution, it is only necessary that the computed Answer Sets cover at least one negative example. We therefore use only the fact instance $ex(negative)$ as representative of any negative example. Given that the hypotheses search space, S_M, is formally represented in the ASP encoding, to identify specific hypothesis solutions for a given set of (positive and negative examples) we use a predicate $active(R_{id})$, where R_{id} is a unique identifier for a rule in S_M. This predicate is added in the body of each rule $R \in S_M$. Rules that are not chosen as optimal inductive solutions will have this condition evaluated to false (and the rule will still be

vacuously satisfied). Inductive solutions will therefore be the set of rules whose corresponding $active(R_{id})$ is true. Formally, given an Answer Set A, the function $meta^{-1}(A) = \{R \in S_M : active(R_{id}) \in A\}$.

Definition 8. *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 every $e^+ = \langle \{li_1, \ldots, li_h\}, \{le_1, \ldots, le_k\} \rangle \in E^+$ the rules*
   - *$ex(ex^+_{id})$*
   - *$\leftarrow$ not $example\_covered(ex^+_{id})$*
   - *$example\_covered(e^+_{id}) \leftarrow e(li_1, ex^+_{id}), \ldots, e(li_h, ex^+_{id}),$ not $e(le_1, ex^+_{id}), \ldots,$ not $e(le_k, ex^+_{id})$*
4. *$meta(E^-)$ includes for every $e^- = \langle \{li_1, \ldots, li_h\}, \{le_1, \ldots, le_k\} \rangle \in E^-$ the rule*
   - *$violating \leftarrow e(li_1, negative), \ldots, e(li_h, negative),$ not $e(le_1, negative), \ldots,$ not $e(le_k, negative)$*
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 6. Recall the task T in Example 4. T^3_{meta} is as follows:

1. $meta(B) = \{e(q, X) \leftarrow e(q, X), ex(X)\}$
2. $meta(S_M) = \{e(p, X) \leftarrow active(a), ex(X); e(r, X) \leftarrow active(b), ex(X);$
   $e(p, X) \leftarrow e(r, X), active(c), ex(X);$ $e(p, X) \leftarrow$ not $e(r, X), active(d), ex(X);$
   $e(r, X) \leftarrow$ not $e(q, X), active(e), e(X)\}$
3. $meta(E^+) = \{example\_covered(1) \leftarrow ex(p, 1); example\_covered(2) \leftarrow ex(p, 2);$
   $\leftarrow$ not $example\_covered(1); \leftarrow$ not $example\_covered(2);$ $ex(1);$ $ex(2)\}$
4. $meta(E^-) = \{violating \leftarrow ex(p, negative), ex(q, negative)\}$
5. $meta(Aux, 3) = \{length(a, 1); length(b, 1); length(c, 2); length(d, 2); length(e, 2);$
   3 #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 $H \in positive\_solutions^n(T)$ 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 8 can be used to generate all the violating solutions. Specifically, given a length n, the ASP program $T^n_{meta} \cup \{\leftarrow$ not $violating;$ $ex(negative)\}$ will have Answer Sets that include $active(R_{id})$ of hypotheses $R \in S_M$ that are violating solutions. This is captured by the following Proposition.

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 $T^n_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(negative)\}$. Then $H \in violating\_solutions^n(T)$ 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 $T^n_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(negative)\}$. Then to convert these solutions into constraints (see Definition 9) 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.

Definition 9. *Let hypothesis $H = \{R_1, \ldots, R_h\}$. We denote with $constraint(H)$ 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.*

Algorithm 1 ILASP

```
procedure ILASP(T)
    solutions = []
    for n = 0; solutions.empty; n++ do
        vs = AS(T^n_meta ∪ {← not violating;  ex(negative).})
        ps = AS(T^n_meta ∪ {constraint(meta^-1(V)) : V ∈ vs})
        solutions = {meta^-1(A) : A ∈ 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 every violating solution H, compute exactly to 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 $P = T^n_{meta} \cup \{constraint(V) : V \in violating\_solutions^n(T)\}$. Then a hypothesis $H \in ILP^n_{LAS}(T)$ if and only if $\exists A \in AS(P)$ such that $H = meta^{-1}(A)$.*

The following theorem states that our algorithm $ILASP$ is sound and complete with respect to the notion of optimal[4] inductive solutions in $ILP^*_{LAS}(T)$. We denote with $ILASP(T)$ the set of inductive solutions computed by ILASP for a given ILP_{LAS} learning 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 a hypothesis of length n and let P be the program $T^n_{meta} \cup \{\leftarrow \text{not } violating;\ \ ex(negative).\}$

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

By Prop. 2, $H \in violating\_solutions^n(T)$ iff $\exists A \in AS(P)$ st $H = meta^{-1}(A)$.
$\Rightarrow H \in violating\_solutions^n(T)$ iff $\exists A \in vs$ st $H = meta^{-1}(A)$.
$\Rightarrow violating\_solutions^n(T) = \{meta^{-1}(A) : A \in vs\}$
$ps = AS(T^n_{meta} \cup \{constraint(meta^{-1}(V)) : V \in vs\})$
$\Rightarrow ps = AS(T^n_{meta} \cup \{constraint(V) : V \in violating\_solutions^n(T)\})$
$\Rightarrow H \in ILP^n_{LAS}(T)$ 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 such that this set is non-empty, $ILASP(T) = ILP^*_{LAS}(T)$. □

5 Application to a Planning Problem

In this section we apply our Learning from Answer Sets 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 on the right 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 it does not know that to go through a locked cell it must first visit the key to that cell. The goal for the agent is to learn an hypothesis for valid moves that will allow it to reach the target position.

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 that illustrate 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 in order to learn the concept of valid move the agent has also to learn a new concept that does not appear in the examples or in the background knowledge. This scenario shows the ability of our learning approach to support predicate invention [11]. Finally, in the third scenario, the environment is non deterministic. The agent then learns a non deterministic notion of valid move.

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

```
unlocked(C, T) :- visited_cell(Key, T), key(C, Key), time(T).
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).
```

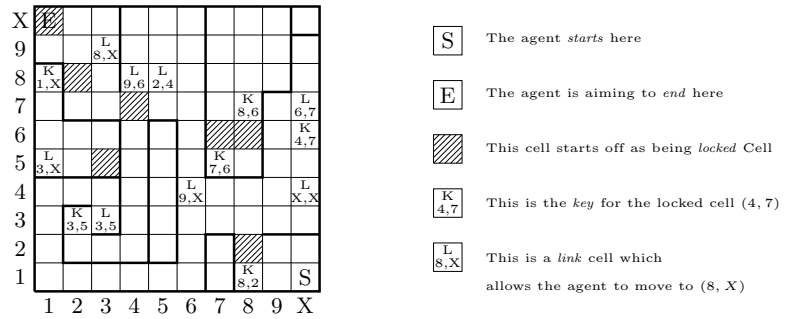

Fig. 2. Cells with diagonal lines are *locked* and the agent must visit the corresponding key before it can enter these cells. *Link* cells allow the agent to jump to the indicated destination cell. The thick black lines represent walls.

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 hypothesis that is consistent with the new information. 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. But at the same time, 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. Similar to the positive example, the set of negative examples is updated every time the agent has to relearn. 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.

Executing our $ILASP$ algorithm, only the shortest hypothesis that is consistent with what the agent has observed so far is computed. As shown in Table 1 $ILASP$ is able to generate the correct solution in 6 learning steps; however, in this scenario it had only to learn one single predicate. In the next scenario 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 the 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 have learned something similar to concept of unlocked cell. However, the agent actually learned a shorter hypothesis:

```
extra(C,T) :- agent_at(C1, V1), link(C1,C).
```

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

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

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 $(1,2)$, $(2,2)$, $(3,2)$).

```
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).
```

In these two scenarios, because of the deterministic environment the agent had only to learn programs with one Answer Set. The next scenario explores what happens when we have a non-deterministic environment.

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. Here we allow aggregates, which extends the search space to include many more rules. We also made a small addition to the background knowledge that combines the concepts of adjacent and wall into a new concept $joined(C1,C2) \leftarrow adjacent(C1,C2)$, 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 a not valid 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 target hypothesis:

```
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 which is closest to our own. We reformulate (but preserve the meaning of) some of the learning tasks that follow to allow for easier comparison with our own.

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 as they only considered a single positive example.

There are other styles of learning task which involve partial interpretations. 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 $B \cup H$, M which extends E. As this definition uses models rather than stable models/Answer Sets, H covers an example E iff $B \cup H \cup E$ is consistent. This is not the case for Learning from Answer Sets. Another approach is *Learning from Interpretation Transition* [7]. The examples here are pairs of interpretations $\langle I, J\rangle$ such that the consequences of $I \cup B \cup H = J$. This relates to the supported model semantics rather than the stable model semantics, making it difficult to compare to our task. However if we needed to construct a similar style of task in ASP, such that if an Answer Set A extends I then it also extends B, we could translate it into an ILP_{LAS} task by constructing, for each literal $L \in J$, a negative example $I \cup \{neg(L)\}$ where $neg(L)$ reverses the sign of L. If we wanted to further stipulate that there is at least one such Answer Set, then we could add I as a positive example.

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. We intend to pursue both. 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: Inductive Logic Programming, pp. 91–97. Springer (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: Logic Programming and Nonmonotonic Reasoning, pp. 169–181. Springer (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.P.: Induction of stable models. In: Inductive Logic Programming, pp. 193–205. Springer (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)