Conflict-driven Inductive Logic Programming

Mark Law
ILASP Limited, UK
(*e-mail:* `mark@ilasp.com`)

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

Abstract

The goal of Inductive Logic Programming (ILP) is to learn a program that explains a set of examples. Until recently, most research on ILP targeted learning Prolog programs. The ILASP system instead learns Answer Set Programs (ASP). Learning such expressive programs widens the applicability of ILP considerably; for example, enabling preference learning, learning common-sense knowledge, including defaults and exceptions, and learning non-deterministic theories.

Early versions of ILASP can be considered *meta-level* ILP approaches, which encode a learning task as a logic program and delegate the search to an ASP solver. More recently, ILASP has shifted towards a new method, inspired by conflict-driven SAT and ASP solvers. The fundamental idea of the approach, called *Conflict-driven ILP* (CDILP), is to iteratively interleave the search for a hypothesis with the generation of constraints which explain why the current hypothesis does not cover a particular example. These *coverage constraints* allow ILASP to rule out not just the current hypothesis, but an entire class of hypotheses that do not satisfy the coverage constraint.

This paper formalises the CDILP approach and presents the ILASP3 and ILASP4 systems for CDILP, which are demonstrated to be more scalable than previous ILASP systems, particularly in the presence of noise.

KEYWORDS: Non-monotonic Inductive Logic Programming, Answer Set Programming Conflict-driven Solving

1 Introduction

Inductive Logic Programming (ILP) (Muggleton 1991) systems aim to find a set of logical rules, called a hypothesis, that, together with some existing background knowledge, explain a set of examples. Unlike most ILP systems, which usually aim to learn Prolog programs, the ILASP (Inductive Learning of Answer Set Programs) systems (Law et al. 2014; Law 2018; Law et al. 2020) can learn Answer Set Programs (ASP), including normal rules, choice rules, disjunctive rules, and hard and weak constraints. ILASP's learning framework has been proven to generalise existing frameworks and systems for learning ASP programs (Law et al. 2018a), such as the brave learning framework (Sakama and Inoue 2009), adopted by almost all previous systems (e.g. XHAIL (Ray 2009), ASPAL (Corapi et al. 2012), ILED (Katzouris et al. 2015), RASPAL (Athakravi et al. 2013)), and the less common cautious learning framework (Sakama and Inoue 2009). Brave systems require the examples to be covered in at least one answer set of the learned program, whereas cautious systems find a program which covers the examples in every answer set. The results in (Law et al. 2018a) show that some ASP programs cannot be learned with either a brave or a cautious approach, and that to learn ASP programs in general, a combination of both brave and cautious reasoning is required. ILASP's learning framework enables this combination, and is capable of learning the full class of ASP programs (Law et al. 2018a). ILASP's generality has allowed it to be applied to a wide range of applications, including event detection (Law et al. 2018b), preference learning (Law et al. 2015), natural language understanding (Chabierski et al. 2017), learning game rules (Cropper et al. 2019), grammar induction (Law et al. 2019) and automata induction (Furelos-Blanco et al. 2020).

Throughout the last few decades, ILP systems have evolved from early bottom-up/top-down learners, such as (Quinlan 1990; Muggleton 1995; Srinivasan 2001), to more modern systems, such as (Cropper and Muggleton 2016; Kaminski et al. 2019; Corapi et al. 2010; Corapi et al. 2012), which take advantage of logic programming systems to solve the task. These recent ILP systems, commonly referred to as *meta-level* systems, work by transforming an ILP learning problem into a meta-level logic program whose solutions can be mapped back to the solutions of the original ILP problem.

At first glance, the earliest ILASP systems (ILASP1 (Law et al. 2014) and ILASP2 (Law et al. 2015)) may seem to be meta-level systems, and they do indeed involve encoding a learning task as a meta-level ASP program; however, they are actually in a more complicated category. Unlike "pure" meta-level systems, the ASP solver is not invoked on a fixed program, and is instead (through the use of multi-shot solving (Gebser et al. 2016)) incrementally invoked on a program that is growing throughout the execution. With each new version, ILASP has shifted further away from pure meta-level approaches, towards a new category of ILP system, which we call *conflict-driven*. Conflict-driven ILP systems, inspired by conflict-driven SAT and ASP solvers, iteratively construct a set of constraints on the solution space that must[1] be satisfied by any inductive solution. In each iteration, the solver finds a program H that satisfies the current constraints, then searches for a *conflict* C, which corresponds to a reason why H is not an (optimal) inductive solution. If none exists, then H is returned; otherwise, C is converted to a new *coverage constraint* which the next programs must satisfy. The process of converting a conflict into a new coverage constraint is called *conflict analysis*.

This paper formalises the notion of Conflict-driven Inductive Logic Programming (CDILP) and presents the ILASP3[2] and ILASP4 systems for CDILP. The CDILP approach (and consequently, both ILASP3 and ILASP4) is proven to be sound and complete (i.e. guaranteed to find an optimal solution for any learning task, provided at least one solution exists), and shown through an evaluation to be significantly

[1] In the case of noisy examples, these are "soft" constraints that should be satisfied, but can be ignored for a penalty.

[2] The ILASP3 system was originally released in 2017 and presented in Mark Law's PhD thesis (Law 2018) using different terminology.

faster than previous ILASP systems on tasks with noisy examples. One of the major advantages of the CDILP approach is that it allows for *constraint propagation*, where a coverage constraint computed for one example is *propagated* to another example. The evaluation in this paper shows that this allows for much more efficient solving of tasks with noisy examples (where examples do not have to be covered, and instead a penalty is paid for each uncovered example), as it allows ILASP to "boost" the penalty incurred by not conforming to a coverage constraint so that a penalty is paid not just for the original example, but also for all of the other examples to which the constraint has been propagated.

We also identify a particular type of learning task called a *non-categorical* learning task, which highlights an inefficiency of ILASP3, which can make the computation of coverage constraints expensive. This is because ILASP3's coverage constraints are constructed to be a complete translation of an example into a constraint that is satisfied if and only if the example is covered. ILASP4 considerably relaxes the notion of a coverage constraint to be a constraint that is *necessary* (but may not be sufficient) for the example to be covered. Such coverage constraints tend to be significantly smaller and cheaper to compute, especially on non-categorical learning tasks.

The CDILP approach (as ILASP3) has already been evaluated on several real datasets and compared with other state-of-the-art ILP systems, which unlike ILASP do not guarantee finding an *optimal* solution of the learning task (in terms of the length of the hypothesis, and the penalties paid for not covering examples), ILASP finds solutions which are on average better quality than those found by the other systems (in terms of the F_1-score on a test set of examples) (Law et al. 2018b). The evaluation in this paper compares the performance of ILASP4 to ILASP3 on several synthetic non-categorical learning tasks, and shows that ILASP4 is often significantly faster than ILASP3.

The CDILP framework is entirely modular, meaning that users of the ILASP system can replace any part of the CDILP approach with their own method, and providing their new method shares the same correctness properties as the original modules in ILASP, their customised CDILP approach will still be sound and complete (and guaranteed to terminate). This customisation is supported in the ILASP implementation through the use of a new Python interface (called PyLASP).

The rest of the paper is structured as follows. Section 2 recalls the necessary background material. Section 3 formalises the notion of Conflict-driven ILP. Section 4 presents several approaches to conflict analysis. Section 5 gives an evaluation of the approach. Finally, Sections 6 and 7 present the related work and conclude the paper.

2 Background

This section introduces the background material that is required to understand the rest of the paper. First, the fundamental Answer Set Programming concepts are recalled, and then the *learning from answer sets* framework used by the ILASP systems is formalised.

2.1 Answer Set Programming

A *disjunctive rule* R is of the form $\mathtt{h_1} \vee \ldots \vee \mathtt{h_m}$ `:-` $\mathtt{b_1}, \ldots, \mathtt{b_n}$, $\mathtt{not\ c_1}, \ldots, \mathtt{not\ c_o}$, where $\{\mathtt{h_1}, \ldots, \mathtt{h_m}\}$, $\{\mathtt{b_1}, \ldots, \mathtt{b_n}\}$ and $\{\mathtt{c_1}, \ldots, \mathtt{c_o}\}$ are sets of atoms denoted $head(R)$, $body^+(R)$ and $body^-(R)$, respectively. A *normal rule* R is a disjunctive rule such that $|head(R)| = 1$. A *definite rule* R is a disjunctive rule such that $|head(R)| = 1$ and $|body^-(R)| = 0$. A *hard constraint* R is a disjunctive rule such that $|head(R)| = 0$. Sets of disjunctive, normal and definite rules are called disjunctive, normal and definite logic programs (respectively).

Given a (First-order) disjunctive logic program P, the Herbrand base (HB_P) is the set of all atoms constructed using constants, functions and predicates in P. The program $ground(P)$ is constructed by replacing each rule with its ground instances (using only atoms from the Herbrand base). An (Herbrand) interpretation I (of P) assigns each element of HB_P to $\top$ or $\bot$, and is usually written as the set of all elements in HB_P that I assigns to $\top$. An interpretation I is a model of P if it satisfies every rule in P; i.e. for each rule $R \in ground(P)$ if $body^+(R) \subseteq I$ and $body^-(R) \cap I = \emptyset$ then $head(R) \cap I \neq \emptyset$. A model of P is *minimal* if no strict subset of P is also a model of P. The *reduct* of P wrt I (denoted P^I) is the program constructed from $ground(P)$ by first removing all rules R such that $body^-(R) \cap I \neq \emptyset$ and then removing all remaining negative body literals from the program. The *answer sets* of P are the interpretations I such that I is a minimal model of P^I. The set of all answer sets of P is denoted $AS(P)$.

There is another way of characterising answer sets, by using *unfounded subsets*. Let P be a disjunctive logic program and I be an interpretation. A subset $U \subseteq I$ is unfounded (w.r.t. P) if there is no rule $R \in ground(P)$ for which the following three conditions all hold: (1) $head(R) \cap I \subseteq U$; (2) $body^+(R) \subseteq I \backslash U$; and (3) $body^-(R) \cap I = \emptyset$. The *answer sets* of a program P are the models of P with no non-empty unfounded subsets w.r.t. P.

Unlike hard constraints in ASP, *weak constraints* do not affect what is, or is not, an answer set of a program P. Hence the above definitions also apply to programs with weak constraints. Weak constraints create an ordering over $AS(P)$ specifying which answer sets are "better" than others. A *weak constraint* is of the form $:\sim \mathtt{b_1}, \ldots, \mathtt{b_n}, \mathtt{not\ c_1}, \ldots, \mathtt{not\ c_m}.[\mathtt{w@l}, \mathtt{t_1}, \ldots, \mathtt{t_k}]$ where $\mathtt{b_1}, \ldots, \mathtt{b_n}, \mathtt{c_1}, \ldots, \mathtt{c_m}$ are atoms, $\mathtt{w}$ and $\mathtt{l}$ are terms specifying the *weight* and the *level*, and $\mathtt{t_1}, \ldots, \mathtt{t_k}$ are terms ($[\mathtt{w@l}, \mathtt{t_1}, \ldots, \mathtt{t_k}]$ is called the *tail* of the weak constraint). At each *priority level* $\mathtt{l}$, the aim is to discard any answer set which does not minimise the sum of the weights of the ground weak constraints (with level $\mathtt{l}$) whose bodies are true. The higher levels are minimised first. 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 $(\mathtt{w}, \mathtt{l}, \mathtt{t_1}, \ldots, \mathtt{t_k})$ for which there is some $:\sim \mathtt{b_1}, \ldots, \mathtt{b_n}, \mathtt{not\ c_1}, \ldots,$ $\mathtt{not\ c_m}.[\mathtt{w@l}, \mathtt{t_1}, \ldots, \mathtt{t_k}]$ in the grounding of P such that A satisfies $\mathtt{b_1}, \ldots, \mathtt{b_n},$ $\mathtt{not\ c_1}, \ldots, \mathtt{not\ c_m}$.

Unless otherwise stated, in this paper the term ASP program is used to mean a program consisting of a finite set of disjunctive rules and weak constraints[3]. The semantics of weak constraints (Calimeri et al. 2013) are defined as follows. For each level l, $P^l_A = \sum_{(w,l,t_1,\ldots,t_k) \in weak(P,A)} w$. For $A_1, A_2 \in AS(P)$, A_1 *dominates* A_2 (written $A_1 \succ_P A_2$) iff $\exists l$ such that $P^l_{A_1} < P^l_{A_2}$ and $\forall m > l, P^m_{A_1} = P^m_{A_2}$. An answer set $A \in AS(P)$ is *optimal* if it is not dominated by any $A_2 \in AS(P)$.

This paper uses the following notation to descibe the preference relationship between a pair of answer sets A_1 and A_2, using six binary comparison operators $\{<, >, \leq, \geq, =, \neq\}$.

- $A_1 <_P A_2$ iff $A_1 \succ_P A_2$;
- $A_1 >_P A_2$ iff $A_2 \succ_P A_1$;
- $A_1 \leq_P A_2$ iff $A_2 \not\succ_P A_1$;
- $A_1 \geq_P A_2$ iff $A_1 \not\succ_P A_2$;
- $A_1 =_P A_2$ iff $A_1 \not\succ_P A_2$ and $A_2 \not\succ_P A_1$;
- $A_1 \neq_P A_2$ iff $A_1 \succ_P A_2$ or $A_2 \succ_P A_1$.

Let WC be a set of weak constraints. The weak constraints in WC are *independent* if there are no two weak constraints W_1 and W_2 in WC with ground instances W_1^g and W_2^g such that the tail of W_1^g is equal to the tail of W_2^g. In this paper, we only consider programs whose weak constraints are independent.

2.2 Learning from Answer Sets

The Learning from Answer Sets framework, introduced in (Law et al. 2014), is targeted at learning ASP programs. The basic framework has been extended several times, allowing learning weak constraints (Law et al. 2015), learning from context-dependent examples (Law et al. 2016) and learning from noisy examples (Law et al. 2018b). This section presents the most general version of the learning framework, ILP^{noise}_{LOAS}, which is used in this paper.

Examples in ILP^{noise}_{LOAS} come in two forms: Context-dependent Partial Interpretations (CDPIs); and Context-dependent Ordering Examples (CDOEs). CDPIs specify what should or should not be an answer set of the learned program, and CDOEs specify which answer sets of the learned program should be preferred to which other answer sets of the learned program. A *partial interpretation* e_{pi} is a pair of sets of atoms $\langle e^{inc}, e^{exc} \rangle$ called the inclusions and the exclusions, respectively. An interpretation I is extended by e_{pi} if and only if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. A *Context-dependent Partial Interpretation* is a pair $\langle e_{pi}, e_{ctx} \rangle$ where e_{pi} is a partial interpretation and e_{ctx} (the *context* of e) is a disjunctive logic program. A program P is said to *accept* e if there is at least one answer set A of $P \cup e_{ctx}$ that extends e_{pi} – such an A is called an *accepting answer set* of e w.r.t. P, written $A \in AAS(e, P)$.

[3] The ILASP systems support a wider range of ASP programs, including choice rules and conditional literals, but we omit these concepts for simplicity.

Example 1
Consider the program P, with the following two rules:

```
heads(V1) :- coin(V1), not tails(V1).
tails(V1) :- coin(V1), not heads(V1).
```

- P accepts $e = \langle\langle\{\texttt{heads(c1)}\}, \{\texttt{tails(c1)}\}\rangle, \{\texttt{coin(c1).}\}\rangle$. The only accepting answer set of e w.r.t. P is $\{\texttt{heads(c1)}, \texttt{coin(c1)}\}$.
- P accepts $e = \langle\langle\{\texttt{heads(c1)}\}, \{\texttt{tails(c1)}\}\rangle, \{\texttt{coin(c1). coin(c2).}\}\rangle$. The two accepting answer sets of e w.r.t. P are $\{\texttt{heads(c1)}, \texttt{heads(c1)}, \texttt{coin(c1)}, \texttt{coin(c2)}\}$ and $\{\texttt{heads(c1)}, \texttt{tails(c1)}, \texttt{coin(c1)}, \texttt{coin(c2)}\}$.
- P does not accept $e = \langle\langle\{\texttt{heads(c1)}, \texttt{tails(c1)}\}, \emptyset\rangle, \{\texttt{coin(c1).}\}\rangle$.
- P does not accept $e = \langle\langle\{\texttt{heads(c1)}\}, \{\texttt{tails(c1)}\}\rangle, \emptyset\rangle$.

In learning from answer sets tasks, CDPIs are given as either *positive* (resp. *negative*) examples, which should (resp. should not) be accepted by the learned program.

Ordering Examples. Positive and negative examples can be used to learn any ASP program consisting of normal rules, choice rules and hard constraints.[4] As positive and negative examples can only express what should or should not be an answer set of the learned program, they cannot be used to learn weak constraints, which do not affect what is or is not an answer set. Weak constraints create a preference ordering over the answer sets of a program, so in order to learn them we need to give examples of this preference ordering – i.e. examples of which answer sets should be preferred to which other answer sets. These *ordering examples* come in two forms: *brave orderings*, which express that at least one pair of accepting answer sets for a pair of positive examples is ordered in a particular way; and *cautious orderings*, which express that every such pair of answer sets should be ordered in that way.

A *context-dependent ordering example* (CDOE) o is a tuple $\langle e^1, e^2, \prec\rangle$, where e^1 and e^2 are CDPIs and $\prec$ is a binary comparison operator ($<$, $>$, $=$, $\leq$, $\geq$ or $\neq$). A pair of interpretations $\langle I_1, I_2\rangle$ is said to be an *accepting pair of answer sets* of o w.r.t. a program P if all of the following conditions hold: (i) $I_1 \in AAS(e^1, P)$; (ii) $I_2 \in AAS(e^2, P)$; and (iii) $I_1 \prec_P I_2$. A program P is said to *bravely respect* o if there is at least one accepting pair of answer sets of o w.r.t. P. P is said to *cautiously respect* o if there is no accepting pair of answer sets of $\langle e^1, e^2, \prec^{-1}\rangle$ w.r.t. P (where $<^{-1}$ is $\geq$, $>^{-1}$ is $\leq$, $\leq^{-1}$ is $>$, $\geq^{-1}$ is $<$, $=^{-1}$ is $\neq$ and $\neq^{-1}$ is $=$). In other words, P bravely (resp. cautiously) respects o if *at least one* (resp. *every*) pair of answer sets extending the two CDPIs is ordered correctly (w.r.t. $\prec_P$).

Example 2
Consider a scenario in which a user is planning journeys from one location to another. All journeys consist of several legs, in which the user may take various modes of transport. Other known attributes of the journey legs are the distance of

[4] This result holds, up to strong equivalence, which means that given any such ASP program P, it is possible to learn a program that is strongly equivalent to P (Law et al. 2018a).

the leg, and the crime rating of the area (which ranges from 0 – no crime – to 5 – extremely high). By offering the user various journey options, and observing their choices, we can use ILASP to learn the preferences the user is using to make such choices. The options a user could take can be represented using CDPIs. Four such examples are shown below. Note that the first argument of the example is a unique identifier for the example. This identifier is optional, but is needed when expressing ordering examples.

$e_1 = \langle\langle\emptyset, \emptyset\rangle, \{$
  leg_mode(1, walk).
  leg_crime_rating(1, 2).
  leg_distance(1, 500).
  leg_mode(2, bus).
  leg_crime_rating(2, 4).
  leg_distance(2, 3000).
$\}\rangle$

$e_2 = \langle\langle\emptyset, \emptyset\rangle, \{$
  leg_mode(1, bus).
  leg_crime_rating(1, 2).
  leg_distance(1, 4000).
  leg_mode(2, walk).
  leg_crime_rating(2, 5).
  leg_distance(2, 1000).
$\}\rangle$

$e_3 = \langle\langle\emptyset, \emptyset\rangle, \{$
  leg_mode(1, bus).
  leg_crime_rating(1, 2).
  leg_distance(1, 400).
  leg_mode(2, bus).
  leg_crime_rating(2, 4).
  leg_distance(2, 3000).
$\}\rangle$

$e_4 = \langle\langle\emptyset, \emptyset\rangle, \{$
  leg_mode(1, bus).
  leg_crime_rating(1, 5).
  leg_distance(1, 2000).
  leg_mode(2, bus).
  leg_crime_rating(2, 1).
  leg_distance(2, 2000).
$\}\rangle$

Consider the program P, with the following three weak constraints.

```
:~ leg_mode(L, walk), leg_crime_rating(L, C), C > 3.[1@3, L, C]
:~ leg_mode(L, bus).[1@2, L]
:~ leg_mode(L, walk), leg_distance(L, D).[D@1, L, D]
```

These weak constraints represent that the user's top priority is to minimise the number of legs of the journey in which the user must walk through an area with a high crime rating; their next priority is to minimise the number of buses the user must take; and finally, their lowest priority is to minimise the total walking distance of their journey.

- P bravely (and cautiously) respects $\langle e_1, e_2, <\rangle$.
- P bravely (and cautiously) respects $\langle e_3, e_4, =\rangle$.

Note that in this scenario there is always a single answer set of $B \cup H \cup C$ for each of the contexts C, meaning that there is no difference between bravely respecting and cautiously respecting an ordering. When $B \cup H \cup C$ may have multiple answer sets, the distinction is important, and cautious orderings are much stronger than brave orderings, expressing that the preference ordering holds universally over all pairs of accepting answer sets of the positive examples.

Noisy examples. In settings where all examples are correctly labelled (i.e. there is no noise), ILP systems search for a hypothesis that covers all of the examples. Many systems search for the optimal such hypothesis – this is usually defined as the hypothesis minimising the number of literals in H ($|H|$). In real settings, examples are often not guaranteed to be correctly labelled. In these cases, many ILP systems (including ILASP) assign a penalty to each example, which is the *cost* of not covering the example. A CDPI (resp. CDOE) e can be upgraded to a *weighted CDPI* (resp. *weighted CDOE*) by adding a penalty e_{pen}, which is either a positive integer or ∞, and a unique identifier, e_{id}, for the example.

Learning Task. Definition 1 formalises the ILP_{LOAS}^{noise} *learning task*, which is the input of the ILASP systems. A *rule space* S_M (often called a hypothesis space and characterised by a *mode bias*[5] M) is a finite set of disjunctive rules and weak constraints, defining the space of programs that are allowed to be learned. Given a rule space S_M, a *hypothesis* H is any subset of S_M.

Definition 1
An ILP_{LOAS}^{noise} task T is a tuple of the form $\langle B, S_M, \langle E^+, E^-, O^b, O^c\rangle\rangle$, where B is an ASP program called the *background knowledge*, S_M is a rule space, E^+ and E^- are (finite) sets of weighted CDPIs and O^b and O^c are (finite) sets of weighted CDOEs. Given a hypothesis $H \subseteq S_M$,

1. $\mathcal{U}(H, T)$ is the set consisting of: (a) all positive examples $e \in E^+$ such that $B \cup H$ does not accept e; (b) all negative examples $e \in E^-$ such that $B \cup H$ accepts e; (c) all brave ordering examples $o \in O^b$ such that $B \cup H$ does not bravely respect o; and (d) all cautious ordering examples $o \in O^c$ such that $B \cup H$ does not cautiously respect o.
2. the score of H, denoted as $\mathcal{S}(H, T)$, is the sum $|H| + \sum_{e \in \mathcal{U}(H,T)} e_{pen}$.
3. H is an inductive solution of T (written $H \in ILP_{LOAS}^{noise}(T)$) if and only if $\mathcal{S}(H, T)$ is finite.
4. H is an *optimal inductive solution* of T (written $H \in {}^*ILP_{LOAS}^{noise}(T)$) if and only if $\mathcal{S}(H, T)$ is finite and $\nexists H' \subseteq S_M$ such that $\mathcal{S}(H, T) > \mathcal{S}(H', T)$.

We say that a learning task T is *categorical* if for each CDPI e in T, there is at most one interpretation I such that there is a hypothesis $H \subseteq S_M$ s.t. $I \in AAS(e, B \cup H)$. A learning task which is not categorical is called *non-categorical*.

3 Conflict-driven Inductive Logic Programming

In this section, we present ILASP's *Conflict-driven ILP* algorithm. Over the course of the execution, a set of *coverage constraints* is constructed. Roughly speaking, these are boolean constraints over the rules that a hypothesis must contain to cover a particular example; for example, they may specify that a hypothesis must contain

[5] We omit details of mode biases, as they are not necessary to understand the rest of this paper. For details of the mode biases supported in ILASP, please see the ILASP manual at `https://doc.ilasp.com/`.

at least one of a particular set of rules and none of another set of rules. The coverage constraints supported by ILASP are formalised by Definitions 2 and 3. Throughout the rest of the paper, we assume $T = \langle B, S_M, E \rangle$ to be an ILP^{noise}_{LOAS} learning task. We also assume that every rule R in S_M has a unique identifier, written R_{id}.

Definition 2
Let S_M be a rule space. A *coverage formula* over S_M takes one of the following forms:

- R_{id}, for some $R \in S_M$.
- $\Sigma(w_1 : R^1_{id}; \ldots ; w_n : R^n_{id}) \prec w$, where $R_1, \ldots, R_n \in S_M$, $w_1, \ldots, w_n, w \in \mathbb{Z}$ and $\prec \in \{<, >, \leq, \geq, =, \neq\}$.
- $\neg F$, where F is a coverage formula over S_M.
- $F_1 \vee \ldots \vee F_n$, where $F_1, \ldots, F_n$ are coverage formulas over S_M.
- $F_1 \wedge \ldots \wedge F_n$, where $F_1, \ldots, F_n$ are coverage formulas over S_M.

The semantics of coverage formulas are defined as follows. Given a hypothesis H:

- R_{id} accepts H if and only if $R \in H$.
- $\Sigma(w_1 : R^1_{id}; \ldots ; w_n : R^n_{id}) \prec w$ accepts H if and only if $\left(\sum_{i \in [1,n], R^i \in H} w_i \right) \prec w$.
- $\neg F$ accepts H if and only if F does not accept H.
- $F_1 \vee \ldots \vee F_n$ accepts H if and only if $\exists i \in [1, n]$ s.t. F_i accepts H.
- $F_1 \wedge \ldots \wedge F_n$ accepts H if and only if $\forall i \in [1, n]$ s.t. F_i accepts H.

Example 3
Consider the following rule space S_M:

h^1 : `heads.`
h^2 : `tails.`
h^3 : `heads:- tails.`
h^4 : `tails:- heads.`
h^5 : `heads:- not tails.`
h^6 : `tails:- not heads.`

Consider the three coverage formulas: $F_1 = (h^1_{id} \vee h^5_{id}) \wedge \neg h^2_{id} \wedge \neg h^4_{id} \wedge \neg h^6_{id}$, $F_2 = (h^1_{id} \vee h^3_{id}) \wedge (h^2_{id} \vee h^4_{id}) \wedge (h^1_{id} \vee h^2_{id})$ and $F_3 = (h^1_{id} \vee h^3_{id}) \wedge (h^2_{id} \vee h^4_{id}) \wedge (h^1_{id} \vee h^2_{id}) \wedge \Sigma(1 : h^1; 1 : h^2; 1 : h^3; 1 : h^4) < 3$.

- F_1 accepts the following hypotheses: $\{h^1\}$, $\{h^5\}$, $\{h^1, h^5\}$, $\{h^1, h^3\}$, $\{h^3, h^5\}$ and $\{h^1, h^3, h^5\}$. No other hypothesis is accepted by F_1.
- Let s be any subset of $\{h^1, h^2, h^3, h^4, h^5, h^6\}$. F_2 accepts the hypothesis s if it contains at least one of the following sets of rules: $\{h^1, h^2\}$, $\{h^1, h^4\}$ and $\{h^2, h^3\}$. No other hypothesis is accepted by F_2.
- Let s be any subset of $\{h^5, h^6\}$. F_3 accepts the following hypotheses: $\{h^1, h^2\} \cup s$, $\{h^1, h^4\} \cup s$ and $\{h^3, h^2\} \cup s$. No other hypothesis is accepted by F_2.

Definition 3
A *coverage constraint* is a pair $\langle e, F \rangle$, where e is an example in E and F is a coverage formula, such that for any $H \subseteq S_M$, if e is covered then H respects F.

Example 4
Consider the rule space S_M from Example 3, an empty background knowledge B, and the (positive) CDPI example $e = \langle\langle\{\texttt{heads}\}, \emptyset\rangle, \emptyset\rangle$. For e to be covered by a hypothesis $H \subseteq S_M$, $AS(H)$ must contain at least one of the answer sets $A_1 = \{\texttt{heads}\}$ or $A_2 = \{\texttt{heads}, \texttt{tails}\}$. The two coverage formulas F_1 and F_2 from Example 3 are respected by exactly those hypotheses of which A_1 and A_2 (respectively) are answer sets. Hence, $\langle e, F_1 \vee F_2\rangle$ is a coverage constraint. Note that respecting $F_1 \vee F_2$ is both necessary and sufficient for a hypothesis to cover e. Definition 3 does not require coverage constraints to have a coverage formula which is sufficient for the example to be covered (only necessary). A key difference between ILASP3 and ILASP4 (formalised in the next section) is that the coverage formulas computed by ILASP3 are guaranteed to be sufficient, whereas those computed by ILASP4 are not. An example of a coverage constraint containing a formula that is necessary but not sufficient is $\langle e, h^1_{id} \vee h^3_{id} \vee h^5_{id}\rangle$.

During the CDILP procedure, a set of coverage constraints *CC* is *solved*, yielding: (1) a hypothesis H which is optimal w.r.t. *CC*; (2) a set of examples U which are known not to be covered by H; and (3) a score s which gives the score of H, according to the coverage constraints in *CC*. These three elements, H, U and s form a *solve result*, which is formalised by the following definition.

Definition 4
Let *CC* be a set of coverage constraints. A *solve result* is a tuple $\langle H, U, s\rangle$, such that:

1. $H \subseteq S_M$;
2. U is the set of examples e (of any type) in E for which there is at least one coverage constraint $\langle e, F\rangle$ such that F does not accept H;
3. $s = |H| + \sum\limits_{u \in U} u_{pen}$;
4. s is finite.

A solve result $\langle H, U, s\rangle$ is said to be *optimal* if there is no solve result $\langle H', U', s'\rangle$ such that $s > s'$.

Theorem 1 shows that for any solve result $\langle H, U, s\rangle$, every example in U is not covered by H and s is a lowerbound for the score of H.

Theorem 1
Let *CC* be a set of coverage constraints. For any solve result $\langle H, U, s\rangle$, $U \subseteq \mathcal{U}(H, T)$ and $s \leq \mathcal{S}(H, T)$.

Proof
Let u be an arbitrary example in U. There must be at least one coverage constraint $\langle u, F\rangle \in CC$ such that H is not accepted by F. By definition of $\langle u, F\rangle$ being a coverage constraint, any hypothesis that is not accepted by F cannot cover u; hence, H does not cover u, and thus $u \in \mathcal{U}(H, T)$. Therefore, $U \subseteq \mathcal{U}(H, T)$. This implies that $\sum\limits_{u \in U} u_{pen} \leq \sum\limits_{u \in \mathcal{U}(H,T)} u_{pen}$, and hence, $s \leq \mathcal{S}(H, T)$. □

Theorem 2 shows that s is equal to the score of H if and only if U is equal to the set of examples not covered by H.

Theorem 2
Let CC be a set of coverage constraints. For any solve result $\langle H, U, s\rangle$, $s = \mathcal{S}(H, T)$ if and only if $U = \mathcal{U}(H, T)$.

Proof
We can show this by demonstrating that $U \neq \mathcal{U}(H, T)$ if and only if $s \neq \mathcal{S}(H, T)$. Assume that $U \neq \mathcal{U}(H, T)$. By Theorem 1, this holds if and only if $U \subset \mathcal{U}(H, T)$, and hence, if and only if $s < \mathcal{S}(H, T)$. By Theorem 1, this holds if and only if $s \neq \mathcal{S}(H, T)$. □

A crucial consequence of Theorems 1 and 2 (formalised by Corollary 1) is that if H is not an optimal inductive solution, then for any optimal solve result containing H there will be at least one counter example to H that is not in U. This means that when a solve result is found such that U contains every example that is not covered by H, H is guaranteed to be an optimal inductive solution of T. This is used as the termination condition for the CDILP procedure in the next section.

Corollary 1
Let CC be a set of coverage constraints. For any optimal solve result $\langle H, U, s\rangle$, such that H is not an optimal solution of T, there is at least one counterexample to H that is not in U.

3.1 The CDILP Procedure

The algorithm presented in this section is a cycle comprised of four steps, illustrated in Figure 1. Step 1, the *hypothesis search*, computes an optimal hypothesis H w.r.t. the current set of coverage constraints. When examples may be noisy (i.e. when they have a finite penalty), some coverage constraints may not be respected by H, and so this first step also computes the set of examples U whose coverage constraints are not respected (i.e. those examples which are known not to be covered). Step 2, the *counterexample search*, finds an example e which is not in U (i.e. an example whose coverage constraints are respected by H) that H does not cover. The existence of such an example e is called a *conflict*, and shows that the coverage constraints are incomplete. The third step, *conflict analysis*, resolves the situation by computing a new coverage constraint for e that is not respected by H. The fourth step, *constraint propagation*, is optional and only useful for noisy tasks. The idea is to check whether the newly computed coverage constraint can also be used for other examples, thus "boosting" the penalty that must be paid by any hypothesis that does not respect the coverage constraint and reducing the number of iterations of the CDILP procedure. The CDILP procedure is formalised by Algorithm 1.

Hypothesis Search. The hypothesis search phase of the CDILP procedure finds an optimal solve result of the current CC if one exists; if none exists, it returns `nil`. This search is performed using Clingo. By default, this process uses Clingo 5's (Gebser et al. 2016) C++ API to enable multi-shot solving (adding any new

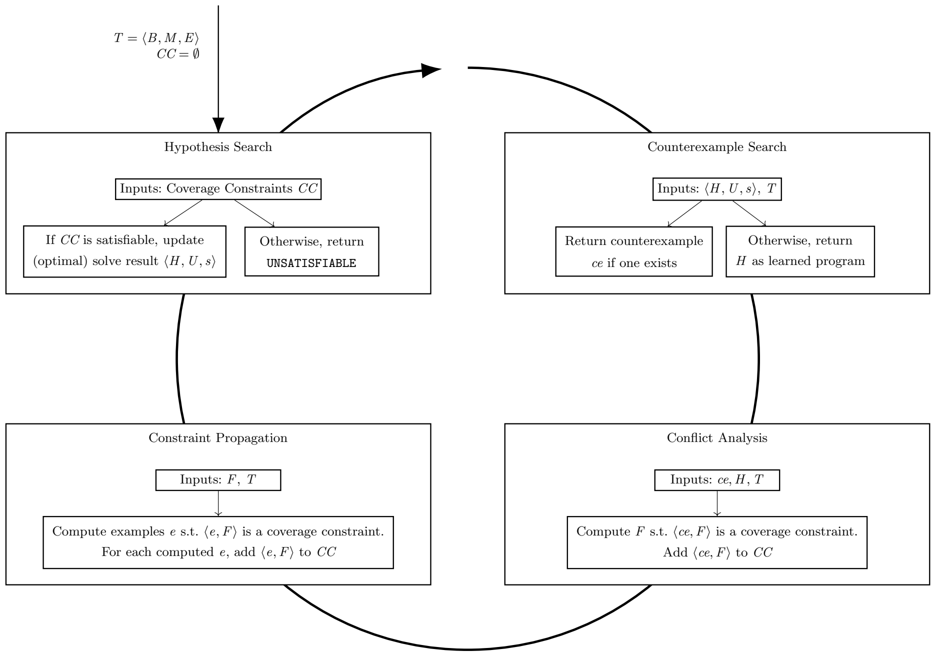

Fig. 1: The ILASP CDILP procedure.

coverage constraints to the program and instructing the solver to continue from where it left off in the previous iteration). This multi-shot solving can be disabled by calling ILASP with the "`--restarts`" flag. For reasons of brevity, the actual ASP encodings are omitted from this paper, but detailed descriptions of very similar ASP encodings can be found in (Law 2018).

Counterexample Search. A counterexample to a solve result $\langle H, U, s\rangle$, is an example e that is not covered by H and is not in U. The existence of such an example proves that the score s is lower than $\mathcal{S}(H, T)$, and hence, H may not be an optimal solution of T. This search is again performed using Clingo, and is identical to the *findRelevantExample* method of ILASP2i (Law et al. 2016) and ILASP3 (Law 2018). If no counterexample exists, then by Corollary 1, H must be an optimal solution of T, and is returned as such; if not, the procedure continues to the conflict analysis step.

Conflict Analysis. Let $\langle H, U, s\rangle$ be the most recent solve result and ce be the most recent counterexample. The goal of this step is to compute a coverage formula F that does not accept H but that must hold for ce to be covered. The coverage constraint $\langle ce, F\rangle$ is then added to CC. This means that if H is computed in a solve result in the hypothesis search phase of a future iteration, then it will be guaranteed to be found with a higher score (including the penalty paid for not covering ce). There are many possible strategies for performing conflict analysis, several of which are presented in the next section and evaluated in Section 5. Beginning with ILASP

Algorithm 1 *CDILP*(T)

```
procedure CDILP(T)
    CC = ∅;
    solve_result = hypothesis_search(CC);
    while solve_result ≠ nil do
        ⟨H, U, s⟩ = solve_result;
        ce = counterexample_search(solve_result, T);
        if ce == nil then
            return H;
        else
            F = conflict_analysis(ce, H, T);
            CC.insert(⟨ce, F⟩)
            prop_egs = propagate_constraints(F, T);
            for each e ∈ prop_egs do
                CC.insert(⟨e, F⟩)
            end for
            solve_result = hypothesis_search(CC);
        end if
    end while
    return UNSATISFIABLE;
end procedure
```

version 4.0.0, the ILASP system allows a user to customise the learning process by providing a Python script (called a *PyLASP* script). Future versions of ILASP will likely contain many strategies, appropriate for different domains and different kinds of learning task. In particular, this allows a user to define their own conflict analysis methods. Provided the conflict analysis method is guaranteed to terminate and compute a coverage constraint whose coverage formula does not accept the most recent hypothesis, the customised CDILP procedure is guaranteed to terminate and return an optimal solution of T (resources permitting). We call such a conflict analysis method *valid*. The three conflict analysis methods presented in the next section are proven to be valid.

Constraint Propagation. The final step is optional (and can be disabled in ILASP with the flag "`-ncp`"). In a task with many examples with low penalties, but for which the optimal solution has a high score, there are likely to be many iterations required before the hypothesis search phase finds an optimal solution. This is because each new coverage constraint only indicates that the next hypothesis computed should either conform to the coverage constraint, or pay a very small penalty. The goal of constraint propagation is to find a set of examples which are guaranteed to not be covered by any hypothesis H that is not accepted by F. For each such example e, $\langle e, F\rangle$ can be added as a coverage constraint (this is called *propagating* the constraint to e). Any solve result containing a hypothesis that does not conform to F must pay the penalty not only for the counterexample ce, but also for every constraint that F was propagated to. In Section 5, it is shown that by lowering the number of iterations required to solve a task, constraint propagation can greatly reduce the overall execution time.

There are two methods of constraint propagation supported in the current version of ILASP. Both were used in ILASP3 and described in detail in (Law 2018) as "implication" and "propagation", respectively. The first is used for positive examples and brave ordering examples, and for each example e searches for a hypothesis that is not accepted by F, but that covers e. If none exists, then the constraint can be propagated to e. The second method, for negative examples, searches for an accepting answer set of e that is guaranteed to be an answer set of $B \cup H \cup e_{ctx}$ for any hypothesis H that is accepted by F. A similar method is possible for propagating constraints to cautious orderings; however, our initial experiments have shown it to be ineffective in practice, as although it does bring down the number of iterations required, it adds more computation time than it saves. Similarly to the conflict analysis phase, users can provide their own strategy for constraint propagation in PyLASP, and in future versions of ILASP will likely have a range of alternative constraint propagation strategies built in.

3.1.1 Proof of Correctness

Theorem 3 proves the correctness of the CDILP approach. The proof of the theorem assumes that the conflict analysis method is valid (which is proven for the three conflict analsysis methods presented in the next section). A *well-formed* task has finite number of examples, and for each example context C, the program $B \cup S_M \cup C$ has a finite grounding. The theorem shows that the CDILP approach is guaranteed to terminate, and is both sound and complete w.r.t. the optimal solutions of a task; i.e. any hypothesis returned is guaranteed to be an optimal solution and if at least one solution exists, then CDILP is guaranteed to return an optimal solution.

Theorem 3
For any ILP^{noise}_{LOAS} well-formed task T, $CDILP(T)$ is guaranteed to terminate and return an optimal solution of T if T is satisfiable, and return `UNSATISFIABLE` otherwise.

Proof
As there is a finite number of hypotheses $H \subseteq S_M$ and a finite number of sets of examples U from the task T, there is a finite number of solve results. Hence, to demonstrate that $CDILP(T)$ terminates, it suffices to show that the hypothesis search phase cannot produce the same solve result in two iterations (demonstrating that there must be a finite number of iterations). Consider an arbitrary iteration with a solve result $\langle H, U, s\rangle$. As the conflict analysis method is valid, it must produce a coverage formula F such that H is not accepted by F. As $\langle u, F\rangle$ is added as a coverage constraint, for any future solve result $\langle H', U', s'\rangle$ such that $H' = H$, U' must contain u. Thus, as $u \notin U$, the solve result $\langle H, U, s\rangle$ cannot be produced by the hypothesis search phase of any future iteration. Hence, $CDILP(T)$ must terminate.

Assume that T is satisfiable. Then there must be at least one optimal solution H^* of T. For any set of coverage constraints CC, $\langle H^*, U_{CC}, s_{CC} \rangle$ is a solve result of CC (where U_{CC} is the set of examples u for which there is at least one $\langle u, F \rangle$ such that F does not accept H and s_{CC} is the corresponding score). Hence, $CDILP(T)$ cannot return UNSATISFIABLE. As $CDILP(T)$ terminates, this means that it must return a hypothesis H. Assume for contradiction that $CDILP(T)$ returns a suboptimal solution H'. In the final iteration, the solve result $\langle H', U', s' \rangle$ must be such that $U' = \mathcal{U}(H, T)$, and hence (by Theorem 2), $s' = \mathcal{S}(H', T)$. As H' is suboptimal, $\mathcal{S}(H', T) > \mathcal{S}(H^*, T)$. Hence $\langle H', U', s' \rangle$ could not have been an optimal solve result for CC, as s_{CC} must be lower than s'. This contradiction proves that $CDILP(T)$ must return an optimal solution of T.

It remains to show that if T is unsatisfiable, $CDILP(T)$ returns UNSATISFIABLE. As $CDILP(T)$ terminates, it suffices to show that $CDILP(T)$ cannot return a hypothesis. Assume for contradiction that it does. Then there must be a set of coverage constraints CC and a solve result $\langle H, U, s \rangle$ such that $U = \mathcal{U}(H, T)$. As s is finite, by Theorem 2, this shows that H has a finite score. Hence, H is a solution of T, which contradicts that T is unsatisfiable. Hence, if T is unsatisfiable, $CDILP(T)$ must return UNSATISFIABLE. □

3.2 Comparison to previous ILASP systems

ILASP1 and ILASP2 both encode the search for an inductive solution as a meta-level ASP program. They are both iterative algorithms and use multi-shot solving (Gebser et al. 2016) to add further definitions and constraints to the meta-level program throughout the execution. However, the number of rules in the grounding of the initial program is roughly proportional to the number of rules in the grounding of $B \cup S_M$ (together with the rules in the contexts of each example) multiplied by the number of positive examples and (twice the number of) brave orderings (Law 2018). This means that neither ILASP1 nor ILASP2 scales well w.r.t. the number of examples.

ILASP2i attempts to remedy the scalability issues of ILASP2 by iteratively constructing a set of *relevant* examples. The procedure is similar to CDILP in that it searches (using ILASP2) for a hypothesis that covers the current set of relevant examples, and then searches for a counterexample to the current hypothesis, which is then added to the set of relevant examples before the next hypothesis search. This simple approach allows ILASP2i to scale to tasks with large numbers of examples, providing the final set of relevant examples stays relatively small (Law et al. 2016). However, as ILASP2i uses ILASP2 for the hypothesis search, it is still using a meta-level ASP program which has a grounding that is proportional to the number of relevant examples, meaning that if the number of relevant examples is large, the scalability issues remain.

The CDILP approach defined in this section goes further than ILASP2i in that the hypothesis search phase is now completely separate from the groundings of the rules in the original task. Instead, the program used by the hypothesis search phase only needs to represent the set of (propositional) coverage formulas.

The major advantage of the CDILP approach compared to ILASP2i (as demonstrated by the evaluation in Section 5) is on tasks with noisy examples. Tasks with noisy examples are likely to lead to a large number of relevant examples. This is because if a relevant example has a penalty it does not need to be covered during the hypothesis search phase. It may be required that a large number of "similar" examples are added to the relevant example set before any of the examples are covered. The CDILP approach overcomes this using constraint propagation. When the first example is found, it will be propagated to all "similar" examples which are not covered for the same "reason". In the next iteration the hypothesis search phase will either have to attempt to cover the example, or pay the penalty of all of the similar examples.

4 Conflict Analysis

This section presents two approaches to conflict analysis available in the ILASP system. The first approach, used by the ILASP3 algorithm, works by translating the counterexample into a coverage formula which is both necessary and sufficient for the example to be covered (i.e. the coverage formula accepts exactly those hypotheses which cover the example). Computing such a coverage formula can be extremely expensive, and so ILASP4 uses a second approach that only guarantees finding a formula which is necessary for the example to be covered. This may mean that the same counterexample is found in multiple iterations of the CDILP procedure (which cannot occur in ILASP3), but for each iteration, the conflict analysis phase is usually significantly cheaper. The evaluation in Section 5 demonstrates that although the number of iterations in ILASP4 is likely to be higher than in ILASP3, the overall running time is often much lower.

There is a lot of space between finding the large sufficient and necessary formulas computed by ILASP3 and the shorter necessary formulas computed by ILASP4. At the end of this section, we present a middle ground approach which is also available in the ILASP4 implementation, which often outperforms the other two approaches.

To aid readability, only the methods for computing coverage formulas from CDPIs are presented in the main paper. The methods for computing coverage formulas from CDOEs are given (together with their proofs of correctness) in the appendix. Most proofs of theorems in this section are also given in the appendix rather than the main paper.

4.1 ILASP3: Example Translation

ILASP3 performs conflict analysis by translating a counterexample into a coverage formula which is accepted by exactly those hypotheses which cover the counterexample (i.e. the formula is necessary and sufficient for the example to be covered).

When translating a CDPI e, the coverage formula computed by ILASP3 is a disjunction $D_1 \vee \ldots \vee D_n$, which is constructed iteratively. The intuition is that in the i^{th} iteration the algorithm searches for a hypothesis that accepts e, but which is not accepted by $D_1 \vee \ldots \vee D_{i-1}$. To prove that the example is accepted by such a hypothesis H the algorithm tries to find an answer set of $B \cup e_{ctx} \cup H$ that extends e_{pi}. If such an answer set I exists, a coverage formula D_i is computed (and added to the disjunction) s.t. D_i is accepted by exactly those hypotheses H' s.t. $I \in AS(B \cup e_{ctx} \cup H')$. If no such answer set exists then $F = D_1 \vee \ldots \vee D_{i-1}$ is necessary for the example to be accepted. If the example e is positive, then $\langle e, F\rangle$ is a coverage constraint; if e is negative then $\langle e, \neg F\rangle$ is a coverage constraint.

This incremental computation relies on being able to generate from an interpretation I a coverage formula that is accepted by exactly those hypotheses H such that $I \in AS(B \cup e_{ctx} \cup H)$. A general formalisation of such a coverage formula is given in the following definition. Note that the definition is strongly linked to the unfounded sets definition of answer sets (given in the Section 2), which says that answer sets of a program P are the models of P with no non-empty unfounded subsets.

Definition 5
Let I be an interpretation and e be a CDPI. The *translation* of $\langle I, e\rangle$ (denoted $\mathcal{T}(I, e, T)$) is the coverage formula constructed by taking the conjunction of the following coverage formulas:

1. $\neg R_{id}$ for each $R \in S_M$ such that I is not a model of R.
2. $R^1_{id} \vee \ldots \vee R^n_{id}$ for each subset minimal set of rules $\{R^1, \ldots, R^n\}$ such that there is at least one non-empty unfounded subset of I w.r.t. $B \cup e_{ctx} \cup (S_M \backslash \{R^1, \ldots, R^n\})$.

We write $\mathcal{T}_1(I, e, T)$ and $\mathcal{T}_2(I, e, T)$ to refer to the conjunctions of coverage formulas in (1) and (2), respectively. Note that the empty conjunction is equal to $\top$.

Example 5
Consider an ILP^{noise}_{LOAS} task with background knowledge B and hypothesis space S_M as defined below.

$$B = \left\{ \begin{array}{l} \texttt{p:- not q.} \\ \texttt{q:- not p.} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} h^1: & \texttt{r:- t.} \\ h^2: & \texttt{t:- q.} \\ h^3: & \texttt{r.} \end{array} \right\}$$

Let e be a CDPI such that $e_{pi} = \langle\{\texttt{r}\}, \emptyset\rangle$, and $e_{ctx} = \emptyset$. Consider the three interpretations $I_1 = \{\texttt{p}, \texttt{r}\}$, $I_2 = \{\texttt{q}, \texttt{r}\}$ and $I_3 = \{\texttt{q}, \texttt{r}, \texttt{t}\}$. The translations are as follows:

- $\mathcal{T}(I_1, e, T) = h^3_{id}$.
- $\mathcal{T}(I_2, e, T) = \neg h^2_{id} \wedge h^3_{id}$.
- $\mathcal{T}(I_3, e, T) = h^2_{id} \wedge (h^1_{id} \vee h^3_{id})$.

The following theorem shows that the translation of an interpretation I (w.r.t. an example e) is a coverage formula that captures the class of hypotheses H for which I is an accepting answer set of $B \cup H$ w.r.t. e.

Theorem 4
Let e be a CDPI and I be a model of $B \cup e_{ctx}$. For any hypothesis $H \subseteq S_M$, $I \in AS(B \cup H \cup e_{ctx})$ accepts e if and only if the translation of $\langle I, e \rangle$ accepts H.

Proof
Assume $I \in AS(B \cup H \cup e_{ctx})$.

$\Leftrightarrow$ I is a model of $B \cup H \cup e_{ctx}$ and there are no non-empty subsets of I w.r.t. $B \cup H \cup e_{ctx}$

$\Leftrightarrow$ I is a model of H and there are no non-empty subsets of I w.r.t. $B \cup H \cup e_{ctx}$ (as I is a model of $B \cup e_{ctx}$)

$\Leftrightarrow$ I is a model of H and for each set of rules $\{R^1, \ldots, R^n\}$ s.t. there is a non-empty unfounded subset of I w.r.t. $B \cup e_{ctx} \cup (S_M \backslash \{R^1, \ldots, R^n\})$, $H \cap \{R^1, \ldots, R^n\} \neq \emptyset$.

$\Leftrightarrow$ I is a model of H and for each subset minimal set of rules $\{R^1, \ldots, R^n\}$ s.t. there is a non-empty unfounded subset of I w.r.t. $B \cup e_{ctx} \cup (S_M \backslash \{R^1, \ldots, R^n\})$, $H \cap \{R^1, \ldots, R^n\} \neq \emptyset$.

$\Leftrightarrow$ the translation of $\langle I, e \rangle$ accepts H.

$\square$

Algorithm 2 formalises the *translateCDPI* procedure used by ILASP3 to translate a CDPI into a coverage formula. It takes an iterative approach, translating one interpretation per iteration until the coverage formula F becomes necessary for e to be accepted.

Algorithm 2 *translateCDPI*(T, e)

procedure TRANSLATECDPI(T, e)
  $F = \bot$;
  while $\exists I$, $\exists H \subseteq S_M$ s.t. F does not accept H and $I \in AAS(e, B \cup H)$ **do**
    Fix an arbitrary such I
    $F = F \vee \mathcal{T}(I, e, T)$;
  end while
  return F;
end procedure

Example 6
Reconsider the ILP^{noise}_{LOAS} task from Example 5. Let e be the CDPI $\langle\langle\{\mathtt{r}\}, \emptyset\rangle, \emptyset\rangle$. The *translateCDPI*$(T, e)$ procedure starts by initialising F as $\bot$. In the first iteration, one interpretation I that satisfies the condition of the while loop is $\{\mathtt{p}, \mathtt{r}\}$, with $H = \{h^3\}$. If this I is selected in the first iteration then F becomes h^3_{id}. In the second iteration, one possible interpretation that satisfies the condition of the while loop is $I = \{\mathtt{p}, \mathtt{q}, \mathtt{r}\}$, with $H = \{h^1, h^2\}$. If this I is selected, then F becomes $h^3_{id} \vee (h^2_{id} \wedge (h^1_{id} \vee h^3_{id}))$. There are no further I's that can satisfy the condition of the while loop, so F is returned.

Example 6 demonstrates that it is not always necessary to translate every interpretation that could be an accepting answer set of e. In fact, it is often possible to translate a much smaller set of interpretations. Theorem 5 proves that *translateCDPI* is guaranteed to terminate, and Theorem 6 proves that the returned coverage formula (negated in the case of a negative example) can be used as a coverage constraint for e. As the theorem shows that there is no hypothesis that is accepted by the coverage formula that does not cover e, when the algorithm is used as a method for conflict analysis on the most recent hypothesis H (which does not cover e), the coverage formula must not accept H. Hence, the approach in this section is a valid method for conflict analysis.

Theorem 5
Let e be a CDPI. The procedure *translateCDPI*(T, e) is guaranteed to terminate.

Theorem 6
Let e be a CDPI and let F be the coverage formula returned by *translateCDPI*(T, e).

1. If $e \in E^+$, the pair $\langle e, F \rangle$ is a coverage constraint and there is no hypothesis accepted by F that covers e.
2. If $e \in E^-$, the pair $\langle e, \neg F \rangle$ is a coverage constraint and there is no hypothesis accepted by $\neg F$ that covers e.

4.2 ILASP4: Computing a Necessary Constraint

Note that for categorical learning tasks, the example translation approach used by ILASP3 is guaranteed to result in a disjunction with only a single disjunct, meaning that the while loop in translateCDPI will only have a single iteration. In practice, however, many tasks are non-categorical (for example, tasks requiring learning recursive definitions or requiring predicate invention, or tasks where the learned program has multiple answer sets). On non-categorical tasks the example translation approach used by ILASP3 can be extremely expensive, and lead to very large coverage formulas. This is because ILASP3 computes a coverage formula which is both necessary and sufficient for the example to be covered (i.e. an exact translation of the example). In fact, the CDILP procedure defined in the previous section does not require a coverage constraint to be sufficient for an example to be covered. It only requires it to be necessary and for the coverage formula to not accept the current hypothesis. In this section, we present methods for extracting much smaller constraints from counterexamples for positive and negative CDPI examples. Note that unlike in ILASP3, there is a different method for positive and negative examples. This is precisely because the coverage constraints are no longer guaranteed to be sufficient – the negation of the coverage formula which is sufficient for a CDPI to be accepted is necessary for the CDPI to not be accepted, but the negation of the coverage formula which is only necessary for a CDPI to be accepted may not be necessary for the CDPI to not be accepted. The following subsections formalise the methods for computing necessary coverage formulas from positive and negative CDPI examples.

4.2.1 Necessary Constraints for Positive Examples

When the translation approach in the previous section is called, the CDILP procedure has a current hypothesis H and a counterexample CDPI e that is not covered. The translation approach ignores the current hypothesis H and computes a disjunction F where each disjunct is a coverage formula which is sufficient and necessary for a particular interpretation to be an answer set (of $B \cup H \cup e_{ctx}$). As each disjunct is a conjunction of conditions that are specific to that answer set, for the formula F to be necessary to accept e, F is may contain a large number of disjuncts. In fact, if in each iteration we were to pick only a single condition from the conjunction as the disjunct, we would still end up with a necessary condition for e to be covered (any formula constructed in this way is guaranteed to be a consequence of F), and is likely to contain fewer disjuncts and therefore be computed faster; however, in general such a method is not valid – as explained in Section 3, a valid method must always produce a coverage formula which does not accept the most recent hypothesis (i.e. it should explain why e is a counterexample to H). In order to guarantee that the method is valid, the approach in this section uses the most recent hypothesis to select the condition for each disjunct. As each disjunct is sufficient for e to be covered at least one of its conjuncts must not be satisfied by H (or H would cover the example e). By selecting exactly one such conjunct from each disjunct (rather than the full formula as in ILASP3), F then becomes a much smaller formula, which is essentially an explanation of why H does not cover e.

The approach is formalised by Algorithm 3. Note that in general the approach could lead to many different coverage formulas, depending on which conjunct is selected in each iteration. The approach in Algorithm 3 favours first selecting conditions from those interpretations I which can become answer sets if rules are added to H, and then only when no such interpretations exist, selecting interpretations which can become answer sets by removing rules from H. This is because it is more costly to add rules (as it increases the hypothesis length) than to take them away, so coverage formulas which are constructed in this way are likely to "move the search on" faster and lead to fewer iterations of the CDILP procedure. However, this is only supported by intuition and initial experimentation, and there may well be other selection strategies which perform better – future versions of ILASP will likely include many such selection strategies.

Example 7
Consider an ILP^{noise}_{LOAS} task with background knowledge B and hypothesis space S_M as defined below.

$$B = \left\{ \begin{array}{l} \texttt{p:- not q.} \\ \texttt{q:- not p.} \\ \texttt{t:- s.} \end{array} \right\} \qquad S_M = \left\{ \begin{array}{ll} h^1: & \texttt{t:- p.} \\ h^2: & \texttt{t:- q.} \\ h^3: & \texttt{r.} \\ h^4: & \texttt{s.} \end{array} \right\}$$

Let e be a CDPI such that $e_{pi} = \langle \{\texttt{t}, \texttt{r}\}, \emptyset \rangle$, and $e_{ctx} = \emptyset$. And let $H = \{h^3\}$. If the translation approach of ILASP3 were to be used, the coverage formula $(\neg h^4_{id} \wedge h^3_{id} \wedge$

Algorithm 3 *CDPI2NC*(T, e)

procedure CDPI2NC(T, H, e)
  $F = \bot$;
  while $\exists I \in M(H)$, $\exists H' \subseteq S_M$ s.t. F does not accept H' and $I \in AAS(e, B \cup H')$ **do**
    Fix an arbitrary such I;
    Let d be an arbitrary conjunct of $\mathcal{T}_2(I, e, T)$ that does not accept H;
    $F = F \vee d$;
  end while
  while $\exists I$, $\exists H' \subseteq S_M$ s.t. F does not accept H' and $I \in AAS(e, B \cup H')$ **do**
    Fix an arbitrary such I;
    $F = F \vee \mathcal{T}_1(I, e, T)$;
  end while
  return F;
end procedure

$(h^1_{id} \vee h^2_{id})) \vee (h^3_{id} \wedge h^4_{id})$ would be returned. The CDPI2NC approach instead returns $h^1_{id} \vee h^2_{id} \vee h^4_{id}$. Note that this formula is a consequence of the formula computed by the translation approach, and is therefore necessary for e to be covered; however, it is not sufficient, meaning that in a future iteration, the hypothesis $H' = \{h^1\}$ may be learned, leading to e again being a counterexample and the additional coverage formula h^3_{id} being computed.

The following two theorems show that the CDPI2NC algorithm is guaranteed to terminate and is a valid method for computing a coverage constraint for a positive CDPI.

Theorem 7
Let e be a CDPI in E^+ and $H \subseteq S_M$ be a hypothesis that does not cover e. The procedure *CDPI2NC*(T, H, e) is guaranteed to terminate.

Theorem 8
Let e be a CDPI in E^+ and $H \subseteq S_M$ be a hypothesis that does not cover e. Let F be the coverage formula returned by *CDPI2NC*(T, H, e).

1. F does not accept H.
2. The pair $\langle e, F \rangle$ is a coverage constraint.

4.2.2 Necessary Constraints for Negative Examples

The method for computing a necessary constraint for a negative CDPI e is much simpler than for positive CDPIs. Note that each disjunct in the formula F computed by the translateCDPI method is sufficient but not necessary for the CDPI e to be accepted (i.e. for e to not be covered, as it is a negative example), and therefore its negation is necessary but not sufficient for e to be covered. This means that to compute a necessary constraint for a negative CDPI, we only need to consider a single interpretation. This is formalised by the following theorem. The approach used in ILASP4 computes an arbitrary such coverage formula. Note that this is guaranteed to terminate and the following theorem shows that the method is a valid method for conflict analysis.

Theorem 9

Let e be a CDPI in E^{-} and $H \subseteq S_M$ be a hypothesis that does not cover e. $AAS(e, B \cup H)$ is non-empty and for any $I \in AAS(e, B \cup H)$:

1. $\neg\mathcal{T}(I, e, T)$ does not accept H.
2. $\langle e, \neg\mathcal{T}(I, e, T)\rangle$ is a coverage constraint.

4.3 A middle ground approach

The approaches for computing coverage constraints from a positive CDPI e in ILASP3 and ILASP4 can be viewed as two extremes; one computes a complete translation of e, and the other computes a very specific reason why the current hypothesis does not cover e. The first approach, taken by ILASP3, is likely to lead to very long conflict analysis (and constraint propagation) times in each iteration, but very few iterations of the CDILP algorithm (as no example will ever be a counterexample twice). The second approach, taken by ILASP4, is likely to lead to shorter conflict analysis (and constraint propagation) times, but may also lead to more iterations of the CDILP algorithm because the same counterexample could be found many times. The CDPI2NC algorithm can be tweaked so that it produces a slightly larger constraint, providing a middle ground between the two approaches.

The basic idea of the alternative CDPI2NCb algorithm is that rather than selecting just one conjunct from $\mathcal{T}_2(I, e, T)$, it uses the entire conjunction. Note that this formula is often significantly simpler than the formula generated by the translation approach of ILASP3. Our evaluation (in the next section) shows that ILASP4 with the CDPI2NCb approach to conflict analysis often outperforms both ILASP3 and ILASP4 with the CDPI2NC approach. For the rest of the paper, to distinguish between the two approaches, we refer to ILASP4a and ILASP4b to mean ILASP4 with CDPI2NC and CDPI2NCb, respectively. The CDPI2NCb algorithm is identical to the CDPI2NC algorithm, other than the first while loop.

Algorithm 4 *CDPI2NCb*(T, e)

procedure CDPI2NCb(T, H, e)
    $F = \bot$;
    while $\exists I \in M(H)$, $\exists H' \subseteq S_M$ s.t. F does not accept H' and $I \in AAS(e, B \cup H')$ **do**
        Fix an arbitrary such I;
        $F = F \vee \mathcal{T}_2(I, e, T)$;
    end while
    while $\exists I$, $\exists H' \subseteq S_M$ s.t. F does not accept H' and $I \in AAS(e, B \cup H')$ **do**
        Fix an arbitrary such I;
        $F = F \vee \mathcal{T}_1(I, e, T)$;
    end while
    return F;
end procedure

The following two theorems show that CDPI2NCb is guaranteed to complete and is a valid method for conflict analysis on positive CDPIs.

Theorem 10
Let e be a CDPI in E^+ and $H \subseteq S_M$ be a hypothesis that does not cover e. The procedure *CDPI2NCb*(T, H, e) is guaranteed to terminate.

Theorem 11
Let e be a CDPI in E^+ and $H \subseteq S_M$ be a hypothesis that does not cover e. Let F be the coverage formula returned by *CDPI2NCb*(T, H, e).

1. F does not accept H.
2. The pair $\langle e, F \rangle$ is a coverage constraint.

Discussion. Note that for each of the three approaches to conflict analysis described in this section, on categorical learning tasks the performance is likely to be fairly similar, because there is at most one possible accepting answer set for each CDPI in the task, meaning that each coverage formula is guaranteed to have at most one disjunct. However, on non-categorical learning tasks, ILASP3 can be extremely inefficient, as demonstrated by the evaluation in the next section.

5 Evaluation

This section presents an evaluation of ILASP's conflict-driven approach to ILP. The datasets used in this paper have been previously used to evaluate previous versions of ILASP, including ILASP3, (e.g. in (Law 2018; Law et al. 2018b)). ILASP3 has previously been applied to several real-world datasets, including event detection, sentence chunking and preference learning (Law et al. 2018b). Rather than repeating these experiments here, we direct the reader to (Law et al. 2018b), which also gives a detailed comparison between the performance of ILASP3 and other ILP systems on noisy datasets. In this evaluation, we focus on synthetic datasets which highlight the weaknesses of older ILASP systems, and show how (in particular) ILASP4 has overcome them.

5.1 Comparison between ILASP versions on benchmark tasks

In (Law et al. 2016), ILASP was evaluated on a set of non-noisy benchmark problems, designed to test all functionalities of ILASP at the time (at that time, ILASP was incapable of solving noisy learning tasks). The running times of all incremental versions of ILASP (ILASP1 and ILASP2 are incapable of solving large tasks) on these benchmarks are shown in Table 1.[6]

The first benchmark problem is to learn the definition of whether a graph is Hamiltonian or not (i.e. whether it contains a Hamilton cycle). The background knowledge is empty and each example corresponds to exactly one graph, specifying

[6] All experiments in this paper were run on an Ubuntu 20.04 virtual machine with 8 cores and 16GB of RAM, hosted on a server with a 3.0GHz Intel Xeon Gold 6136 processor, unless otherwise noted. All benchmark tasks in this section are available for download from `http://www.ilasp.com/research`.

Task	$\|S_M\|$	$\|E^+\|$	$\|E^-\|$	$\|O^b\|$	$\|O^c\|$	2i	3	4a	4b
Hamilton	104	100	100	0	0	3.09	64.12	4.51	2.59
Noisy Hamilton A	104	29	31	0	0	237.12	35.67	8.98	5.78
Noisy Hamilton B	104	56	64	0	0	TO	68.31	25.11	21.57
Noisy Hamilton C	104	87	93	0	0	TO	62.98	46.95	43.59
Scheduling (3 day)	180	400	0	110	90	10.14	17.01	3.83	4.25
Scheduling (4 day)	180	400	0	128	72	29.04	36.16	5.17	5.15
Scheduling (5 day)	180	400	0	133	67	53.84	22.63	6.55	6.81
Agent A	531	200	0	0	0	29.39	57.34	42.36	32.86
Agent B	146	50	0	0	0	2.56	1091.98	15.22	11.97
Agent C	160	80	120	0	0	14.24	14.73	12.43	9.27
Agent D	244	172	228	390	0	84.51	63.38	38.63	34.99
Journey	117	386	0	200	0	1.35	2.92	1.54	1.66

Table 1: The running times (in seconds) of various ILASP systems on the set of benchmark problems. TO denotes a timeout (where the time limit was 1800s).

which `node` and `edge` atoms should be true. Positive examples correspond to Hamiltonian graphs, and negative examples correspond to non-Hamiltonian graphs. This is the context-dependent "Hamilton B" setting from (Law et al. 2016). ILASP2i and both versions of ILASP4 perform similarly, but ILASP3 is significantly slower than the other systems. This is because the Hamiltonian learning task is non-categorical and the coverage formulas generated by ILASP3 tend to be large. This experiment was repeated with a "noisy" version of the problem where 5% of the examples were mislabelled (i.e. positive examples were changed to negative examples or negative examples were changed to positive examples). To show the scalability issues with ILASP2i on noisy learning tasks, three versions of the problem were run, with 60, 120 and 180 examples. ILASP2i's execution time rises rapidly as the number of examples grows, and it is unable to solve the last two tasks within the time limit of 30 minutes. ILASP3 and ILASP4 are all able to solve every version of the task in far less than the time limit, with ILASP4b performing best. The remaining benchmarks are drawn from non-noisy datasets, where ILASP2i performs fairly well.

The second benchmark problem is that of learning scheduling preferences, first presented in (Law et al. 2015). In this setting, the goal is to learn an academic's preferences about interview scheduling, encoded as weak constraints. The tasks in this case are over examples with `3x3`, `4x3` and `5x3` timetables, respectively (i.e. three day, four day and five day timetables). In this case, ILASP2i and ILASP3 perform fairly similarly, but both versions of ILASP4 are significantly better either ILASP2i or 3. This task is non-categorical, but in fact as there are only weak constraints in the hypothesis space, for brave orderings ILASP3, ILASP4a and ILASP4b are actually guaranteed to compute the same coverage formula. However, for cautious orderings this is not the case, and the two ILASP4 algorithms will tend to compute shorter formulas than ILASP3 (corresponding to a single pair of answer sets that were ordered incorrectly by the previous hypothesis, rather than all possible pairs of answer sets), although there is still no difference between the two ILASP4 algorithms in this case. This explains the improvement in performance of ILASP4 over ILASP3.

The third setting originates from (Law et al. 2014) and is based on an agent learning the rules of how it is allowed to move within a grid. Agent A requires a hypothesis describing the concept of which moves are valid, given a history of where an agent has been. Examples are of the agent's history of moving through the map and a subset of the moves which were valid/invalid at each time point in its history. Agent B requires a similar hypothesis to be learned, but with the added complexity that an additional concept is required to be invented (and used in the rest of the hypothesis). In Agent C, the hypothesis from Agent A must be learned along with a constraint ruling out histories in which the agent visits a cell twice (not changing the definition of valid move). This requires negative examples to be given, in addition to positive examples. In Agent D, weak constraints must be learned to explain why some traces through the grid are preferred to others. This uses positive, negative and brave ordering examples. Although scenarios A, C and D are technically non-categorical, scenario B causes more of an issue for ILASP3 because of the (related) challenge of predicate invention. The potential to invent new predicates which are unconstrained by the examples means that there are many possible answer sets for each example, which leads ILASP3 to generate extremely long coverage formulas. In this case ILASP3 is nearly two orders of magnitude slower than either version of ILASP4.

The final benchmark is based on a dataset from (Law et al. 2016), in which the goal is to learn a user's journey preferences from examples of which journeys the user prefers over other journeys. This task is categorical and contains only weak constraints. For such tasks, ILASP3, ILASP4a and ILASP4b will compute the same coverage formulas in all cases, so there is not much difference between them – minor details of the different implementations cause the ILASP4 approaches to still be slightly faster. All approaches perform similarly to ILASP2i on this task.

5.2 Comparison between methods for conflict analysis on a synthetic noisy dataset

In (Law et al. 2018b) ILASP3 was evaluated on a synthetic noisy dataset in which the task is to learn the definition of what it means for a graph to be Hamiltonian. This concept requires learning a hypothesis that contains choice rules, recursive rules and hard constraints, and also contains negation as failure. The advantage of using a synthetic dataset is that the amount of noise in the dataset (i.e. the number of examples which are mislabelled) can be controlled when constructing the dataset. This allows us to evaluate ILASP's tolerance to varying amounts of noise. ILASP1, ILASP2 and ILASP2i (although theoretically capable of solving any learning task which can be solved by the later systems) are all incapable of solving large noisy learning tasks in a reasonable amount of time. Therefore, this section only presents

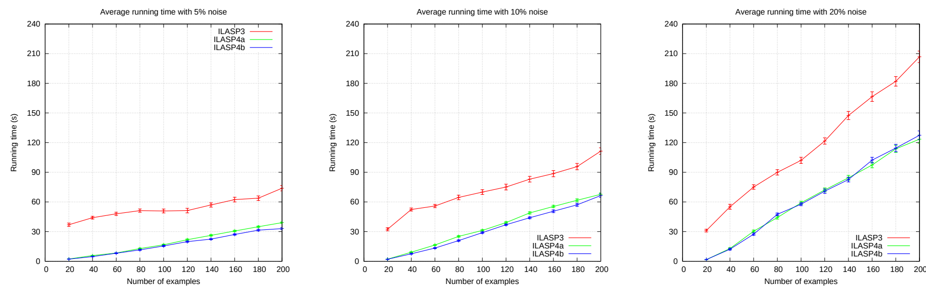

Fig. 2: The average computation time of ILASP3, ILASP4a and ILASP4b for the Hamilton learning task, with varying numbers of examples, with 5, 10 and 20% noise.

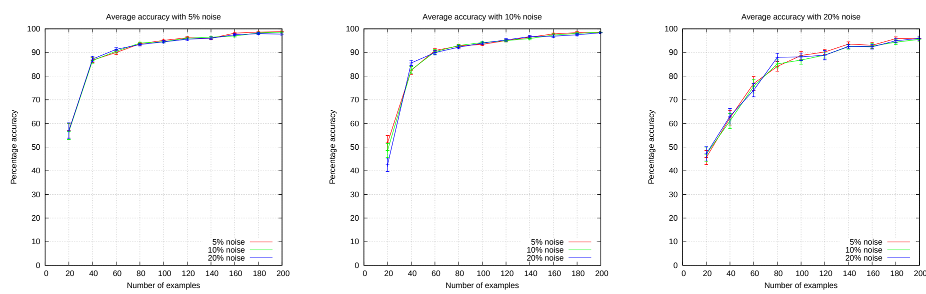

Fig. 3: The average accuracies of ILASP3, ILASP4a and ILASP4b for the Hamilton learning task, with varying numbers of examples, with 5, 10 and 20% noise.

a comparison of the performance of ILASP3 and ILASP4 on the synthetic noisy dataset from (Law et al. 2018b).

For $n = 20, 40, \ldots, 200$, n random graphs of size one to four were generated, half of which were Hamiltonian. The graphs were labelled as either positive or negative, where positive indicates that the graph is Hamiltonian. Three sets of experiments were run, evaluating each ILASP algorithm with 5%, 10% and 20% of the examples being labelled incorrectly. In each experiment, an equal number of Hamiltonian graphs and non-Hamiltonian graphs were randomly generated and 5%, 10% or 20% of the examples were chosen at random to be labelled incorrectly. This set of examples were labelled as positive (resp. negative) if the graph was not (resp. was) Hamiltonian. The remaining examples were labelled correctly (positive if the graph was Hamiltonian; negative if the graph was not Hamiltonian). Figures 2 and 3 show the average running time and accuracy (respectively) of each ILASP version with up to 200 example graphs. Each experiment was repeated 50 times (with different randomly generated examples). In each case, the accuracy was tested by generating a further 1,000 graphs and using the learned hypothesis to classify the graphs as either Hamiltonian or non-Hamiltonian.

The experiments show that each of the three conflict-driven ILASP algorithms (ILASP3, ILASP4a and ILASP4b) achieve the same accuracy on average (this is to

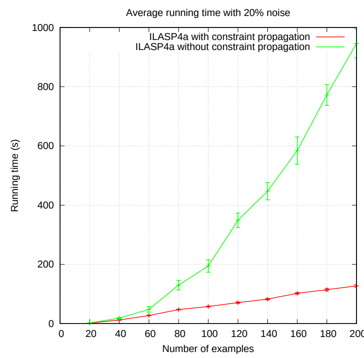

Fig. 4: The average running times of ILASP4a with and without constraint propagation enabled for the Hamilton learning task with 20% noise.

be expected, as each system is guaranteed to find an optimal solution of any task). They each achieve a high accuracy (of well over 90%), even with 20% of the examples labelled incorrectly. A larger percentage of noise means that ILASP requires a larger number of examples to achieve a high accuracy. This is to be expected, as with few examples, the hypothesis is more likely to "overfit" to the noise, or pay the penalty of some non-noisy examples. With large numbers of examples, it is more likely that ignoring some non-noisy examples would mean not covering others, and thus paying a larger penalty. The computation time of each algorithm rises in all three graphs as the number of examples increases. This is because larger numbers of examples are likely to require larger numbers of iterations of the CDILP approach (for each ILASP algorithm). Similarly, more noise is also likely to mean a larger number of iterations. The experiments also show that on average, both ILASP4 approaches perform around the same, with ILASP4b being marginally better than ILASP4a. Both ILASP4 approaches perform significantly better than ILASP3. Note that the results reported for ILASP3 on this experiment are significantly better than those reported in (Law et al. 2018b). This is due to improvements to the overall ILASP implementation (shared by ILASP3 and ILASP4).

The effect of constraint propagation. The final experiment in this section evaluates the benefit of using constraint propagation on noisy learning tasks. The idea of constraint propagation is that although it itself takes additional time, it may decrease the number of iterations of the conflict-driven algorithms, meaning that the overall running time is reduced. Figure 4 shows the difference in running times between ILASP4a with and without constraint propagation enabled on a repeat of the Hamilton 20% noise experiment. Constraint propagation makes a huge difference to the running times demonstrating that this feature of CDILP is a crucial factor in ILASP's scalability over large numbers of noisy examples.

6 Related Work

Learning under the answer set semantics. Traditional approaches to learning under the answer set semantics were broadly split into two categories: *brave* learners (e.g. (Sakama and Inoue 2009; Ray 2009; Corapi et al. 2012; Katzouris et al. 2015; Kazmi et al. 2017)), which aimed to explain a set of (atomic) examples in at least one answer set of the learned program; and *cautious* learners (e.g. (Inoue and Kudoh 1997; Seitzer et al. 2000; Sakama 2000; Sakama and Inoue 2009)), which aimed to explain a set of (atomic) examples in every answer set of the learned program.[7] In general, it is not possible to *distinguish* between two ASP programs (even if they are not strongly equivalent) using either brave or cautious reasoning alone (Law et al. 2018a), meaning that some programs cannot be learned with either brave or cautious induction; for example, no brave induction system is capable of learning constraints – roughly speaking, this is because examples in brave induction only say what *should* be (in) an answer set, so can only incentivise learning programs with new or modified answer sets (compared to the background knowledge on its own), whereas constraints only rule out answer sets. ILASP (Law et al. 2014) was the first system capable of combining brave and cautious reasoning, and (resources permitting) can learn any ASP program[8] up to strong equivalence in ASP (Law et al. 2018a).

FastLAS (Law et al. 2020) is a recent ILP system that solves a restricted version of ILASP's ILP^{noise}_{LOAS} task. Unlike ILASP, it does not enumerate the hypothesis space in full, meaning that it can scale to solve tasks with much larger hypothesis spaces than ILASP. The restrictions on FastLAS mean that it is currently incapable of learning recursive definitions, can only learn programs with a single answer set, and is restricted to observational predicate learning (where every predicate being learned can be directly observed from the examples). Compared to ILASP, these are major restrictions, and work to lift them is ongoing.

Conflict-driven solvers. ILASP's CDILP approach was partially inspired by conflict-driven SAT (Lynce and Marques-Silva 2003) and ASP (Gebser et al. 2007; Gebser et al. 2011; Alviano et al. 2013) solvers, which generate *nogoods* or *learned constraints* (where the term learned should not be confused with the notion of learning in this paper) throughout their execution. These nogoods/learned constraints are essentially reasons why a particular search branch has failed, and allow the solver to rule out any further candidate solutions which fail for the same reason. The coverage formulas in ILASP perform the same function. They are a reason why the most recent hypothesis is not a solution (or, in the case of noisy learning tasks, not as good a solution as it was previously thought to be) and allow ILASP to rule out (or, in the case of noisy learning tasks, penalise) any hypothesis that is not accepted by the coverage formula.

It should be noted that although ILASP3 and ILASP4 are the closest linked

[7] Some of these systems predate the terms brave and cautious induction, which first appeared in (Sakama and Inoue 2009).

[8] Note that some ASP constructs, such as aggregates in the bodies of rules, are not yet supported by the implementation of ILASP, but the abstract algorithms are all capable of learning them.

ILASP systems to these conflict-driven solvers, earlier ILASP systems are also partially conflict-driven. ILASP2 (Law et al. 2015) uses a notion of a *violating reason* to explain why a particular negative example is not covered. A violating reason is an accepting answer set of that example (w.r.t. $B \cup H$). Once a violating reason has been found, not only the current hypothesis, but any hypothesis which shares this violating reason is ruled out. ILASP2i (Law et al. 2016) collects a set of *relevant examples* – a set of examples which were not covered by previous hypotheses – which must be covered by any future hypothesis. However, these older ILASP systems do not extract coverage formulas from the violating reasons/relevant examples, and use an expensive meta-level ASP representation which grows rapidly as the number of violating reasons/relevant examples increases. They also do not have any notion of *constraint propagation*, which is crucial for efficient solving of noisy learning tasks.

Incremental approaches to ILP. Some older ILP systems, such as ALEPH (Srinivasan 2001), Progol (Muggleton 1995) and HAIL (Ray et al. 2003), incrementally consider each positive example in turn, employing a *cover loop*. The idea behind a cover loop is that the algorithm starts with an empty hypothesis H, and in each iteration adds new rules to H such that a single positive example e is covered, and none of the negative examples are covered. Unfortunately, cover loops do not work in a non-monotonic setting because the examples covered in one iteration can be "uncovered" by a later iteration. Worse still, the wrong choice of hypothesis in an early iteration can make another positive example impossible to cover in a later iteration. For this reason, most ILP systems under the answer set semantics (including ILASP1 and ILASP2) tend to be *batch learners*, which consider all examples at once. The CDILP approach in this paper does not attempt to learn a hypothesis incrementally (the hypothesis search starts from scratch in each iteration), but instead builds the set of coverage constraints incrementally. This allows ILASP to avoid the problems of cover loop approaches in a non-monotonic setting, while still overcoming the scalability issues associated with batch learners.

There are two other incremental approaches to ILP under the answer set semantics. ILED (Katzouris et al. 2015), is an incremental version of the XHAIL algorithm, which is specifically targeted at learning Event Calculus theories. ILED's examples are split into *windows*, and ILED incrementally computes a hypothesis through *theory revision* (Wrobel 1996) to cover the examples. In an arbitrary iteration, ILED revises the previous hypothesis H (which is guaranteed to cover the first n examples), to ensure that it covers the first $n + 1$ examples. As the final hypothesis is the outcome of the series of revisions, although each revision may have been optimal, ILED may terminate with a sub-optimal inductive solution. In contrast, every version of ILASP will always terminate with an optimal inductive solution if one exists. The other incremental ILP system under the answer set semantics is RASPAL (Athakravi et al. 2013; Athakravi 2015), which uses an ASPAL-like (Corapi et al. 2012) approach to iteratively revise a hypothesis until it is an optimal inductive solution of a task. RASPAL's incremental approach is successful as it often only needs to consider small parts of the hypothesis space, rather than the full hypothesis space. Unlike ILED and ILASP, however, RASPAL considers the full set of examples when searching for a hypothesis.

Popper (Cropper and Morel 2021) is a recent approach to learning definite programs. It is closely related to CDILP as it also uses an iterative approach where the current hypothesis (if it is not a solution) is used to constrain the future search. However, unlike ILASP, Popper does not extract a coverage formula from the current hypothesis and counterexample, but instead uses the hypothesis itself as a constraint; for example, ruling out any hypothesis that theta-subsumes the current hypothesis. Popper's approach has the advantage that, unlike ILASP, it does not need to enumerate the hypothesis space in full; however, compared to ILASP it is very limited, and does not support negation as failure, choice rules, disjunction, hard or weak constraints, non-observational predicate learning, predicate invention or learning from noisy examples. It is unclear whether the approach of Popper could be extended to overcome these limitations.

ILP approaches to noise. Most ILP systems have been designed for the task of learning from example atoms. In order to search for best hypotheses, such systems normally use a scoring function, defined in terms of the coverage of the examples and the length of the hypothesis (e.g. ALEPH (Srinivasan 2001), Progol (Muggleton 1995), and the implementation of XHAIL (Bragaglia and Ray 2015)). When examples are noisy, this scoring function is sometimes combined with a notion of maximum threshold, and the search is not for an optimal solution that minimises the number of uncovered examples, but for a hypothesis that does not fail to cover more than a defined maximum threshold number of examples (e.g. (Srinivasan 2001; Oblak and Bratko 2010; Athakravi et al. 2013)). In this way, once an acceptable hypothesis (i.e. a hypothesis that covers a sufficient number of examples) is computed the system does not search for a better one. As such, the computational task is simpler, and therefore the time needed to compute a hypothesis is shorter, but the learned hypothesis is not optimal. Furthermore, to guess the "correct" maximum threshold requires some idea of how much noise there is in the given set of examples. For instance, one of the inputs to the HYPER/N (Oblak and Bratko 2010) system is the proportion of noise in the examples. When the proportion of noise is unknown, too small a threshold could result in the learning task being unsatisfiable, or in learning a hypothesis that overfits the data. On the other hand, too high a threshold could result in poor hypothesis accuracy, as the hypothesis may not cover many of the examples. The ILP_{LOAS}^{noise} framework addresses the problem of computing optimal solutions and in doing so does not require any knowledge a priori of the level of noise in the data.

Another difference when compared to many ILP approaches that support noise is that ILP_{LOAS}^{noise} examples contain partial interpretations. In this paper, we do not consider penalising individual atoms within these partial interpretations. This is somewhat similar to what traditional ILP approaches do (it is only the notion of examples that is different in the two approaches). In fact, while penalising individual atoms within partial interpretations would certainly be an interesting avenue for future work, this could be seen as analogous to penalising the arguments of atomic examples in traditional ILP approaches (Law 2018).

XHAIL is a brave induction system that avoids the need to enumerate the entire hypothesis space. XHAIL has three phases: abduction, deduction and induction. In the first phase, XHAIL uses abduction to find a minimal subset of some specified ground atoms. These atoms, or a generalisation of them, will appear in the head of some rule in the hypothesis. The deduction phase determines the set of ground literals which could be added to the body of the rules in the hypothesis. The set of ground rules constructed from these head and body literals is called a kernel set. The final induction phase is used to find a hypothesis which is a generalisation of a subset of the kernel set that proves the examples. The public implementation of XHAIL (Bragaglia and Ray 2015) has been extended to handle noise by setting penalties for the examples similarly to ILP_{LOAS}^{noise}. However, as shown in Example 8 XHAIL is not guaranteed to find an optimal inductive solution of a task.

Example 8
Consider the following noisy task, in the XHAIL input format:

```
p(X) :- q(X, 1), q(X, 2).          #modeh r(+s).
p(X) :- r(X).                      #modeh q(+s2, +t).
s(a).   s(b).   s2(b).             #example not p(a)=50.
t(1).   t(2).                      #example p(b)=100.
```

This corresponds to a hypothesis space that contains two facts $F_1 = \mathtt{r(X)}$, $F_2 = \mathtt{q(X,Y)}$ (in XHAIL, these facts are implicitly "typed", so the first fact, for example, can be thought of as the rule `r(X):- s(X)`). The two examples have penalties 50 and 100 respectively. There are four possible hypotheses: $\emptyset$, F_1, F_2 and $F_1 \cup F_2$, with scores 100, 51, 1 and 52 respectively. XHAIL terminates and returns F_1, which is a suboptimal hypothesis.

The issue is with the first step. The system finds the smallest abductive solution, $\{\mathtt{r(b)}\}$ and as there are no body declarations in the task, the kernel set contains only one rule: `r(b):- s(b).` XHAIL then attempts to generalise to a first order hypothesis that covers the examples. There are two hypotheses which are subsets of a generalisation of `r(b)` (F_1 and $\emptyset$); as F_1 has a lower score than $\emptyset$, XHAIL terminates and returns F_1. The system does not find the abductive solution $\{\mathtt{q(b,1)}, \mathtt{q(b,2)}\}$, which is larger than $\{\mathtt{r(b)}\}$ and is therefore not chosen, even though it would eventually lead to a better solution than $\{\mathtt{r(b)}\}$.

It should be noted that XHAIL does have an *iterative deepening* feature for exploring non-minimal abductive solutions, but in this case using this option XHAIL still returns F_1, even though F_2 is a more optimal hypothesis. Even when iterative deepening is enabled, XHAIL only considers non-minimal abductive solutions if the minimal abductive solutions do not lead to any non-empty inductive solutions.

In comparison to ILASP, in some problem domains, XHAIL is more scalable as it does not start by enumerating the hypothesis space in full. On the other hand, as shown by Example 8, XHAIL is not guaranteed to find the optimal hypothesis, whereas ILASP is. ILASP also solves ILP_{LOAS}^{noise} tasks, whereas XHAIL solves brave induction tasks, which means that due to the generality results in (Law et al. 2018a) ILASP is capable of learning programs which are out of reach for XHAIL no matter what examples are given.

Inspire (Kazmi et al. 2017) is an ILP system based on XHAIL, but with some modifications to aid scalability. The main modification is that some rules are "pruned" from the kernel set before XHAIL's inductive phase. Both XHAIL and Inspire use a meta-level ASP program to perform the inductive phase, and the ground kernel set is generalised into a first order kernel set (using the mode declarations to determine which arguments of which predicates should become variables). Inspire prunes rules which have fewer than Pr instances in the ground kernel set (where Pr is a parameter of Inspire). The intuition is that if a rule is necessary to cover many examples then it is likely to have many ground instances in the kernel. Clearly this is an approximation, so Inspire is not guaranteed to find the optimal hypothesis in the inductive phase. In fact, as XHAIL is not guaranteed to find the optimal inductive solution of the task (as it may pick the "wrong" abductive solution), this means that Inspire may be even further from the optimal. The evaluation in (Law et al. 2018b) demonstrates that on a real dataset, Inspire's approximation leads to lower quality solutions (in terms of the F_1 score on a test set) than the optimal solutions found by ILASP.

7 Conclusion

This paper has presented the Conflict-driven Inductive Logic Programming (CDILP) approach. While the four phases of the CDILP approach are clearly defined at an abstract level, there is a large range of algorithms that could be used for the *conflict analysis* phase. This paper has presented two (extreme) approaches to conflict analysis: the first (used by the ILASP3 system) extracts as much as possible from a counterexample, computing a coverage formula which is accepts by a hypothesis if and only if the hypothesis covers the counterexample; the second (used by the ILASP4 system) extracts much less information from the example and essentially computes an explanation as to why the most recent hypothesis does not cover the counterexample. A third (middle ground) approach is also presented. Our evaluation shows that the selection of conflict analysis approach is crucial to the performance of the system and that although the second and third approaches used by ILASP4 may result in more iterations of the CDILP process than in ILASP3, because each iteration tends to be much shorter, both versions of ILASP4 can significantly outperform ILASP3, especially for a particular type of *non-categorical* learning task, identified in the evaluation.

The evaluation has also demonstrated that the CDILP approach is robust to high proportions of noisy examples in a learning task, and that the constraint propagation phase of CDILP is crucial to achieving this robustness. Constraint propagation allows ILASP to essentially "boost" the penalty associated with ignoring a coverage constraint, by expressing that not only the counterexample associated with the coverage constraint will be left uncovered, but also every example to which the constraint has been propagated.

There is still much scope for improvement, and future work on ILASP will include developing new (possibly domain-dependent) approaches to conflict analysis. The new PyLASP feature of ILASP4 also allows users to potentially implement customised approaches to conflict analysis, by injecting a Python implementation of their conflict analysis method into ILASP.

References

Alviano, M., Dodaro, C., Faber, W., Leone, N., and Ricca, F. 2013. WASP: A native ASP solver based on constraint learning. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer, 54–66.

Athakravi, D. 2015. Inductive logic programming using bounded hypothesis space. Ph.D. thesis, Imperial College London.

Athakravi, D., Corapi, D., Broda, K., and Russo, A. 2013. Learning through hypothesis refinement using answer set programming. In *International Conference on Inductive Logic Programming*. Springer, 31–46.

Bragaglia, S. and Ray, O. 2015. Nonmonotonic learning in large biological networks. In *Inductive Logic Programming*. Springer, 33–48.

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. `https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.0.pdf`.

Chabierski, P., Russo, A., Law, M., and Broda, K. 2017. Machine comprehension of text using combinatory categorial grammar and answer set programs. CEUR Workshop Proceedings.

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.

Cropper, A., Evans, R., and Law, M. 2019. Inductive general game playing. *Machine Learning*, 1–42.

Cropper, A. and Morel, R. 2021. Learning programs by learning from failures. *Machine Learning (to appear)*.

Cropper, A. and Muggleton, S. H. 2016. Metagol system.

Furelos-Blanco, D., Law, M., Russo, A., Broda, K., and Jonsson, A. 2020. Induction of subgoal automata for reinforcement learning. In *AAAI*. Association for the Advancement of Artificial Intelligence.

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.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Wanko, P. 2016. Theory solving made easy with Clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. 2007. Conflict-driven answer set solving. In *IJCAI*. Vol. 7. 386–392.

Inoue, K. and Kudoh, Y. 1997. Learning extended logic programs. In *IJCAI (1)*. 176–181.

Kaminski, T., Eiter, T., and Inoue, K. 2019. Meta-interpretive learning using hex-programs. In *IJCAI*. 6186–6190.

Katzouris, N., Artikis, A., and Paliouras, G. 2015. Incremental learning of event definitions with inductive logic programming. *Machine Learning 100,* 2-3, 555–585.

Kazmi, M., Schüller, P., and Saygin, Y. 2017. Improving scalability of inductive logic programming via pruning and best-effort optimisation. *Expert Systems with Applications 87*, 291–303.

Law, M. 2018. Inductive learning of answer set programs. Ph.D. thesis, Imperial College London.

Law, M., Russo, A., Bertino, E., Broda, K., and Lobo, J. 2019. Representing and learning grammars in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2919–2928.

Law, M., Russo, A., Bertino, E., Broda, K., and Lobo, J. 2020. FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria. In *AAAI*. Association for the Advancement of Artificial Intelligence.

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. 2015. Learning weak constraints in answer set programming. *Theory and Practice of Logic Programming 15,* 4-5, 511–525.

Law, M., Russo, A., and Broda, K. 2016. Iterative learning of answer set programs from context dependent examples. *Theory and Practice of Logic Programming 16,* 5-6, 834–848.

Law, M., Russo, A., and Broda, K. 2018a. The complexity and generality of learning answer set programs. *Artificial Intelligence 259*, 110–146.

Law, M., Russo, A., and Broda, K. 2018b. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*.

Law, M., Russo, A., and Broda, K. 2020. The ILASP system for inductive learning of answer set programs. *The Association for Logic Programming Newsletter*.

Lynce, I. and Marques-Silva, J. 2003. The effect of nogood recording in DPLL-CBJ SAT algorithms. In *Recent Advances in Constraints*. Springer, 144–158.

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

Muggleton, S. 1995. Inverse entailment and progol. *New generation computing 13,* 3-4, 245–286.

Oblak, A. and Bratko, I. 2010. Learning from noisy data using a non-covering ILP algorithm. In *International Conference on Inductive Logic Programming*. Springer, 190–197.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine learning 5,* 3, 239–266.

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

Ray, O., Broda, K., and Russo, A. 2003. Hybrid abductive inductive learning: A generalisation of progol. In *Inductive Logic Programming*. Springer, 311–328.

Sakama, C. 2000. Inverse entailment in nonmonotonic logic programs. In *ILP*. Springer, 209–224.

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

Seitzer, J., Buckley, J. P., and Pan, Y. 2000. Inded: A distributed knowledge-based learning system. *IEEE Intelligent Systems and their Applications 15,* 5, 38–46.

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

WROBEL, S. 1996. First order theory refinement. *Advances in inductive logic programming 32*, 14–33.

Appendix A: Conflict analysis for CDOEs

Translating a CDOE

Similarly to the translation of a CDPI described in the main paper, ILASP3 translates CDOEs by iteratively computing a coverage formula. Before introducing the method for translation, it is necessary to introduce some additional notation, and a way of computing a coverage formula which can be used to determine whether a pair of interpretations are correctly ordered.

Definition 6
Let P be an ASP program and I_1 and I_2 be interpretations. For any integer l, the *optimisation difference* between I_1 and I_2 at l with respect to P (denoted $\Delta_l^P(I_1, I_2)$) is equal to $P_l^{I_1} - P_l^{I_2}$.

Definition 7
Let A_1 and A_2 be a pair of interpretations, $\prec$ be a binary comparison operator and $[l_1, \ldots, l_n]$ be the list of priority levels in $B \cup S_M$ (in descending order). For any $l \in [l_1, \ldots, l_n]$, $\omega(T, A_1, A_2, l, \prec)$ is the coverage formula $\Sigma(\Delta_l^{R^1}(A_1, A_2) : R_{id}^1; \ldots; \Delta_l^{R^n}(A_1, A_2) : R_{id}^n) \prec \Delta_l^B(A_1, A_2)$, where $\{R^1, \ldots, R^n\} = S_M$.

Furthermore, $\omega(T, A_1, A_2, \prec)$ is the disjunction $F_1 \vee \ldots \vee F_n$, where for each $i \in [1, n]$, $F_i = \omega(T, A_1, A_2, l_i, \prec) \wedge \omega(T, A_1, A_2, l_1, =) \wedge \ldots \wedge \omega(T, A_1, A_2, l_{i-1}, =)$.

Theorem 12
Let o be a CDOE, $\prec$ be a binary comparison operator and A_1 and A_2 be interpretations. A hypothesis $H \subseteq S_M$ satisfies $\omega(T, A_1, A_2, \prec)$ if and only if $A_1 \prec_{B \cup H} A_2$.

Algorithm 5 *translateCDOE*$(T, e_1, e_2, \prec)$

procedure TRANSLATECDOE$(T, e_1, e_2, \prec)$
  $F = \bot$;
  while $\exists \langle I_1, I_2 \rangle$ st $\exists H \subseteq S_M$ st F does not accept H, $I_1 \in AAS(e_1, B \cup H)$, $I_2 \in AAS(e_2, B \cup H)$ and $I_1 \prec_{B \cup H} I_2$ **do**
    Fix an arbitrary such $\langle I_1, I_2 \rangle$
    $F = F \vee (\mathcal{T}(I_1, e_1, T) \wedge \mathcal{T}(I_2, e_2, T) \wedge \omega(T, I_1, I_2, \prec))$;
  end while
  return F;
end procedure

The following two theorems show that the translateCDOE procedure can be used to compute coverage constraints for CDOEs. Specifically, they show that the method is guaranteed to terminate and is a valid method for conflict analysis on brave and cautious orderings (where the translation of a cautious ordering involves taking the negation of a coverage formula, similarly to negative examples in the main paper).

Theorem 13
Let e_1 and e_2 be CDPIs and $\prec$ be a binary comparison operator. The procedure *translateCDOE*$(T, e_1, e_2, \prec)$ is guaranteed to terminate.

Theorem 14
Let $o = \langle e_1, e_2, o_{op} \rangle$ be a CDOE.

1. Let F be the coverage formula returned by *translateCDOE*(T, e_1, e_2, o_{op}). If $o \in O^b$, the pair $\langle o, F \rangle$ is a coverage constraint and there is no hypothesis accepted by F that covers o.
2. Let F be the coverage formula returned by *translateCDOE*$(T, e_1, e_2, o_{op}^{-1})$. If $o \in O^c$, the pair $\langle o, \neg F \rangle$ is a coverage constraint and there is no hypothesis accepted by $\neg F$ that covers o.

Necessary Constraints for Brave Ordering Examples

Similarly to the approach taken for positive and negative CDPIs in the main paper, ILASP4 computes a coverage formula which is necessary but not sufficient to cover brave and cautious CDOE examples. This section formalises the approach for brave orderings and the next section formalises the approach for cautious orderings. The algorithm in this section is similar to the CDPI2NC approach in the main paper, but adapted for brave orderings. Again, in each iteration only a single condition from the full translation of the ordering example is selected. Just as is the case for the CDPI2NC algorithm, the CDOE2NC algorithm (Algorithm 6) is guaranteed to produce a coverage formula which does not accept the most recent hypothesis but is necessary for the brave ordering example to be covered.

The following two theorems demonstrate that the CDOE2NC algorithm is guaranteed to terminate and can be used as part of a valid method for conflict analysis on brave ordering examples.

Theorem 15
Let $o = \langle e_1, e_2, \prec \rangle$ be a CDOE in O^b and $H \subseteq S_M$ be a hypothesis that does not cover o. The procedure *CDOE2NC*$(T, H, e_1, e_2, \prec)$ is guaranteed to terminate.

Theorem 16
Let $o = \langle e_1, e_2, \prec \rangle$ be a CDOE in O^b and $H \subseteq S_M$ be a hypothesis that does not cover o. Let F be the coverage formula returned by *CDOE2NC*$(T, H, e_1, e_2, \prec)$.

1. F does not accept H.
2. The pair $\langle o, F \rangle$ is a coverage constraint.

Algorithm 6 *CDOE2NC*$(T, H, e_1, e_2, \prec)$

procedure CDOE2NC$(T, H, e_1, e_2, \prec)$
    $F = \bot$;
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H)$, $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H)$, $I_2 \in AAS(e_2, B \cup H')$ and $I_1 \prec_{B \cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        $F = F \vee \omega(T, I_1, I_2, \prec)$;
    end while
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H')$, $I_1, I_2 \in M(H)$ and $I_1 \prec_{B \cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        Let d be an arbitrary conjunct of $\mathcal{T}_2(I_1, e_1, T)$ or $\mathcal{T}_2(I_2, e_2, T)$ that does not accept H;
        $F = F \vee d$;
    end while
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H')$ and $I_1 \prec_{B \cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        $F = F \vee (\mathcal{T}_1(I_1, e_1, T) \wedge \mathcal{T}_1(I_2, e_2, T))$;
    end while
    return F;
end procedure

Necessary Constraints for Cautious Ordering Examples

Similarly to the ILASP4 approach to conflict analysis for negative examples, a coverage constraint can be computed for a cautious ordering example by finding a single pair of interpretations that are ordered incorrectly by the current hypothesis. For a hypothesis to cover the ordering example, it must either not accept one of the two interpretations as an answer set, or it must order them correctly. The coverage formula expressing these three possibilities is formalised and proven to be correct by the following theorem. The computation of this formula is guaranteed to terminate, and the theorem shows that using this formula is a valid method for conflict analysis on cautious ordering examples.

Theorem 17

Let $o = \langle e_1, e_2, \prec\rangle$ be a CDOE in O^c and $H \subseteq S_M$ be a hypothesis that does not cover o. There is at least one pair of interpretations $\langle I_1, I_2\rangle$ such that $I_1 \in AAS(e_1, B \cup H)$, $I_2 \in AAS(e_2, B \cup H)$ and $I_1 \prec^{-1}_{B \cup H} I_2$:

1. $\neg\mathcal{T}(I_1, e_1, T) \vee \neg\mathcal{T}(I_2, e_2, T) \vee \omega(T, I_1, I_2, \prec)$ does not accept H.
2. $\langle o, \neg\mathcal{T}(I_1, e_1, T) \vee \neg\mathcal{T}(I_2, e_2, T) \vee \omega(T, I_1, I_2, \prec)\rangle$ is a coverage constraint.

A middle ground approach for brave orderings

Similarly to the approach taken in the main paper for positive CDPIs, there is also a “middle ground” approach for brave ordering examples. Just as in the CDPI2NCb method, the changes made to the CDOE2NC method to generate the CDOE2NCb are very small and only involve changing one of the while loops (in this case, the second while loop). The ILASP4b method used in the evaluation uses this CDOE2NCb method rather than the CDOE2NC method.

Algorithm 7 *CDOE2NCb*$(T, H, e_1, e_2, \prec)$ – extract

procedure CDOE2NCb$(T, H, e_1, e_2, \prec)$
    $F = \bot$;
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H)$, $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H)$, $I_2 \in AAS(e_2, B \cup H')$ and $I_1 \prec_{B\cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        $F = F \vee \omega(T, I_1, I_2, \prec)$;
    end while
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H')$, $I_1, I_2 \in M(H)$ and $I_1 \prec_{B\cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        $F = F \vee (\mathcal{T}_2(I_1, e_1, T) \wedge \mathcal{T}_2(I_2, e_2, T))$;
    end while
    while $\exists\langle I_1, I_2\rangle$ s.t. $\exists H' \subseteq S_M$ st F does not accept H', $I_1 \in AAS(e_1, B \cup H')$, $I_2 \in AAS(e_2, B \cup H')$ and $I_1 \prec_{B\cup H'} I_2$ **do**
        Fix an arbitrary such $\langle I_1, I_2\rangle$
        $F = F \vee (\mathcal{T}_1(I_1, e_1, T) \wedge \mathcal{T}_1(I_2, e_2, T))$;
    end while
    return F;
end procedure

The following two theorems demonstrate that the CDOE2NCb algorithm is guaranteed to terminate and can be used as part of a valid method for conflict analysis on brave ordering examples.

Theorem 18
Let $o = \langle e_1, e_2, o_{op}\rangle$ be a CDOE in O^b and $H \subseteq S_M$ be a hypothesis that does not cover o. The procedure *CDOE2NCb*$(T, H, e_1, e_2, \prec)$ is guaranteed to terminate.

Theorem 19
Let $o = \langle e_1, e_2, o_{op}\rangle$ be a CDOE in O^b and $H \subseteq S_M$ be a hypothesis that does not cover o. Let F be the coverage formula returned by *CDOE2NCb*$(T, H, e_1, e_2, \prec)$.

1. F does not accept H.
2. The pair $\langle o, F\rangle$ is a coverage constraint.