Symmetry Breaking for Inductive Logic Programming

Andrew Cropper[1,2*], David M. Cerna[3,4*], Matti Järvisalo[2]
[1]ELLIS Institute Finland
[2]University of Helsinki
[3]Dynatrace Research
[4]Czech Academy of Sciences Institute of Computer Science
andrew.cropper@helsinki.fi, {david.cerna, dcerna}@{dynatrace.com, cs.cas.cz}, matti.jarvisalo@helsinki.fi

Abstract

The goal of inductive logic programming is to search for a hypothesis that generalises training data and background knowledge. The challenge is searching vast hypothesis spaces, which is exacerbated because many logically equivalent hypotheses exist. To address this challenge, we introduce a method to break symmetries in the hypothesis space. We implement our idea in answer set programming. Our experiments on multiple domains, including visual reasoning and game playing, show that our approach can reduce solving times from over an hour to 17 seconds.

Code — https://github.com/logicand-learning-lab/aaai26-symbreak

Extended version — https://arxiv.org/pdf/2508.06263

1 Introduction

Inductive logic programming (ILP) is a form of machine learning (Muggleton 1991; Cropper and Dumancic 2022). The goal is to search a hypothesis space for a hypothesis (a set of rules) that generalises given training examples and background knowledge (BK).

To illustrate ILP, consider the inductive reasoning game Zendo. In this game, one player, the teacher, creates a secret rule that describes structures. The other players, the students, try to discover the secret rule by building structures. The teacher marks whether structures follow or break the rule. The first student to correctly guess the rule wins. For instance, for the positive and negative examples shown in Figure 1, a possible rule is *"there is a small blue piece"*.

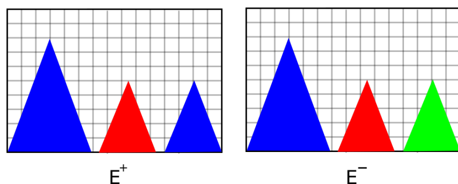

Figure 1: Positive (E^+) and negative (E^-) Zendo examples.

*These authors contributed equally.

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Given these examples and BK about the structures, an ILP system can induce a rule such as:

zendo(A) $\leftarrow$ *piece(A,B), size(B,C), blue(B), small(C)*

The fundamental challenge in ILP is searching large hypothesis spaces. Recent approaches tackle this challenge by delegating the search to an off-the-shelf solver, such as a Boolean satisfiability solver (Ahlgren and Yuen 2013; Hocquette et al. 2024) or an answer set programming (ASP) solver (Law, Russo, and Broda 2014; Kaminski, Eiter, and Inoue 2019; Cropper and Morel 2021; Evans et al. 2021).

One problem for recent approaches is symmetries in the hypothesis space where many logically equivalent hypotheses exist. For instance, consider these two Zendo rules:

r_1 = *zendo(A)* $\leftarrow$ *piece(A,B), size(B,C), blue(B), small(C)*
r_2 = *zendo(A)* $\leftarrow$ *piece(A,C), size(C,B), blue(C), small(B)*

Both rules can be in the hypothesis space. However, the rules are syntactic variants when considering theta-substitution (Plotkin 1971; Lloyd 2012). For instance, $r_1 = r_2\theta$ where $\theta = \{C \mapsto B, B \mapsto C\}$. However, determining whether two rules are symmetrical (syntactic variants) is hard. For instance, theta-substitution is in general NP-complete (Garey and Johnson 1979).

In this paper, we address this symmetry challenge. We first define the *body-variant* problem: determining whether two rules only differ by the naming of body-only variables, such as r_1 and r_2 above. We show that this problem is graph-isomorphism hard (Köbler, Schöning, and Torán 1993).

Due to the hardness of the body-variant problem, we introduce a sound, albeit incomplete, symmetry-breaking approach. Our idea exploits an ordering of variables in a rule. If a body literal l does not contain a variable x, but contains variables both larger and smaller than x according to the variable ordering, then x must occur in a literal lexicographically smaller than l. In other words, any gaps in the occurrence of variables with respect to an ordering must be justified by lexicographically smaller literals. We only order literals with arity greater than one because we assume that rules do not contain singleton variables, so the argument of a unary literal must appear elsewhere in the rule.

To illustrate our approach, consider rules r_1 and r_2 above. Assuming that variables are ordered alphabetically, the literal *piece(A,C)* of r_2 has a gap in its arguments because B is

missing. Furthermore, *size(C,B)* contains B but $(A, C) <_{lex} (B, C)$. By renaming C to B and B to C within r_2 the rule r_1 results. Observe that, r_1 contains both $piece(A, B)$ and $size(B, C)$ and $(A, B) <_{lex} (B, C)$. Thus, we can remove r_2 from the hypothesis space and keep r_1.

We implement our symmetry-breaking idea in ASP and demonstrate it with the ASP-based ILP system POPPER, which can learn optimal and recursive hypotheses from noisy data (Cropper and Morel 2021; Hocquette et al. 2024). We show that adding symmetry-breaking constraints to POPPER can drastically reduce solving and learning time, sometimes from over an hour to only 17 seconds.

Contributions and Novelty. The main novelty of this paper is a method for breaking symmetries in ILP. As far as we are aware, there is no existing work on this topic. The impact, which we demonstrate on multiple domains, is vastly reduced solving and learning times. Moreover, this work connects multiple AI areas, notably machine learning and constraint programming. Overall, we contribute the following:

- We define the problem of determining whether two rules are body-variants (Definition 6). We show that this problem is graph isomorphism hard (Proposition 1).
- We describe a symmetry-breaking approach to prune body-variant rules. We prove that this approach is sound and only prunes body-variant rules (Proposition 2).
- We describe an ASP implementation of our encoding that works with the ILP system POPPER.
- We show experimentally on multiple domains, including visual reasoning and game playing, that our approach can reduce solving and learning times by 99%.

2 Related Work

Rule learning. ILP induces rules from data, similar to rule learning methods (Fürnkranz and Kliegr 2015), such as AMIE+ (Galárraga et al. 2015) and RDFRules (Zeman, Kliegr, and Svátek 2021). Rule-mining methods are typically limited to unary and binary relations and require facts as input and operate under an open-world assumption. By contrast, ILP usually operates under a closed-world assumption, supports relations of any arity, and can learn from definite programs as background knowledge.

ILP. Most classical ILP approaches build a hypothesis using a set covering algorithm by adding rules one at a time (Quinlan 1990; Muggleton 1995; Blockeel and De Raedt 1998; Srinivasan 2001). These approaches build rules literal-by-literal using refinement operators to generalise or specialise a rule (Shapiro 1983; Quinlan 1990; Muggleton 1995; De Raedt and Dehaspe 1997; Blockeel and De Raedt 1998; Srinivasan 2001; Tamaddoni-Nezhad and Muggleton 2009). We differ because we do not use refinement operators. Rather, we break symmetries in an ILP approach that frames the ILP problem as a constraint satisfaction problem.

Constraint-based ILP. Many ILP systems frame the ILP problem as a constraint satisfaction problem, such as SAT (Ahlgren and Yuen 2013; Hocquette et al. 2024) and ASP (Corapi, Russo, and Lupu 2011; Law, Russo, and Broda 2014; Schüller and Benz 2018; Kaminski, Eiter, and Inoue 2019; Evans et al. 2021; Bembenek, Greenberg, and Chong 2023). Most of these approaches formulate the ILP problem as rule selection where they precompute every possible rule in the hypothesis space and use a solver to search for a subset that generalises the examples. Because they precompute all possible rules, they cannot learn rules with many literals. By contrast, we build constraints to restrict rule generation without precomputation.

Redundancy in ILP. Symmetry breaking is a form of redundancy elimination. There is much work on avoiding redundancy in ILP, such as when testing hypotheses on training data (Struyf and Blockeel 2003). Raedt and Ramon (2004) check whether a rule has a redundant atom before testing it on examples to avoid a coverage check. Zeng, Patel, and Page (2014) prune rules with syntactic redundancy. For the rule *h(X)* $\leftarrow$ *p(X,Y), p(X,Z)*, they detect that *p(X,Z)* duplicates *p(X,Y)* under the renaming $Z \mapsto Y$, where Z and Y are not in other literals. By contrast, we reduce redundancy through declarative symmetry-breaking constraints.

Symmetry Breaking. Highly symmetric problems, when represented declaratively, can lead solvers such as ASP and SAT to perform redundant search. Lex-leader (Crawford et al. 1996) uses tools for computing graph symmetries (McKay 1981; Junttila and Kaski 2007; Darga et al. 2004) to automatically identify instance-specific symmetries. Detected symmetries are eliminated by introducing lex-leader symmetry-breaking constraints during preprocessing. This approach has been employed in SAT (Aloul, Markov, and Sakallah 2003; Anders, Brenner, and Rattan 2024), ASP (Drescher, Tifrea, and Walsh 2011; Devriendt and Bogaerts 2016), and other declarative approaches. Tarzariol, Gebser, and Schekotihin (2021) use ILP to learn first-order ASP symmetry-breaking constraints from examples of instance-specific constraints. By contrast, we break symmetries in the ILP hypothesis space. Our notion of symmetry differs from conventional domain-level symmetries (Devriendt et al. 2016). In our work, symmetry refers to syntactic variable interchangeability in rule bodies: two rules are symmetric if one can be obtained from the other by renaming variables.

3 Inductive Logic Programming

In this section, we define ILP and describe prerequisite notation. Let $\mathcal{P}$ be a countably infinite set of predicate symbols and $\mathcal{V}$ be a countably infinite set of variables totally well-ordered by $<_{\mathcal{V}}$. A *term* is either a constant or a variable. Every $p \in \mathcal{P}$ has an arity $0 \leq a$ denoted $arity(p)$. An *atom* is of the form $p(t_1, \ldots, t_a)$ where $p \in \mathcal{P}$, $t_1, \ldots, t_a$ are terms. Given an atom $l = p(t_1, \ldots, t_a)$, $sym(l) = p$, $arity(l) = a$, $args(l) = (t_1, \ldots, t_a)$ where $(\ldots)$ is an ordered tuple. We refer to an atom as *ground* if all its arguments are constants and *constant-free* if all its arguments are variables. A literal is an atom or a negated atom. A rule r is of the form $h \leftarrow p_1, \ldots, p_n$ where $h, p_1, \ldots, p_n$ are literals, $head(r) = h$, $body(r) = \{p_1, \ldots, p_n\}$, and $body_{\geq 2}(r)$ is the subset of $body(r)$ containing all literals in r with arity ≥ 2. We will only consider rules that are *head-connected*, i.e. all variables occurring in the head of a rule appear in a literal of the body of the rule. We refer

to a rule as *constant-free* if all of its literals are constant-free. By $vars(l)$ we denote the set of variables in a literal l. The variables of a rule r, denoted $vars(r)$, is defined as $head_vars(r) \cup body_vars(r)$ where $head_vars(r) = vars(head(r))$ and $body_vars(r) = vars(body(r))$.

We formulate our approach in the ILP learning from entailment setting (De Raedt 2008). We define an ILP input:

Definition 1 (**ILP input**). An ILP input is a tuple $(E, B, \mathcal{H})$ where $E = (E^+, E^-)$ is a pair of sets of ground atoms denoting positive (E^+) and negative (E^-) examples, B is background knowledge, and $\mathcal{H}$ is a hypothesis space, i.e a set of possible hypotheses.

We restrict hypotheses and background knowledge to definite programs with the least Herbrand model semantics (Lloyd 2012). We define a cost function:

Definition 2 (**Cost function**). Given an ILP input $(E, B, \mathcal{H})$, a cost function $cost_{E,B} : \mathcal{H} \mapsto \mathbb{N}$ assigns a numerical cost to each hypothesis in $\mathcal{H}$.

We define an *optimal* hypothesis:

Definition 3 (**Optimal hypothesis**). Given an ILP input $(E, B, \mathcal{H})$ and a cost function $cost_{E,B}$, a hypothesis $h \in \mathcal{H}$ is *optimal* with respect to $cost_{E,B}$ when $\forall h' \in \mathcal{H}$, $cost_{E,B}(h) \leq cost_{E,B}(h')$.

4 Complexity of Symmetry Breaking in ILP

Our goal is to find a subset of a hypothesis space that contains at least one optimal hypothesis after symmetry breaking:

Definition 4 (**Hypothesis space reduction problem**). Given an ILP input $(E, B, \mathcal{H})$, the *hypothesis space reduction problem* is to find $\mathcal{H}' \subseteq \mathcal{H}$ such that if $\mathcal{H}$ contains an optimal hypothesis then there exists an optimal hypothesis $h \in \mathcal{H}'$.

We focus on breaking symmetries by removing hypotheses that differ only by renaming variables. We define a *body-variant* rule:

Definition 5 (**Body variant**). A rule r' is a *body-variant* of a rule r if there exists a bijective renaming σ from $body_vars(r)$ to $body_vars(r')$ such that $r\sigma = r'$.

Example 1. Consider the rules in the introduction:
r_1 = *zendo(A) ← piece(A,B), size(B,C), blue(B), small(C)*
r_2 = *zendo(A) ← piece(A,C), size(C,B), blue(C), small(B)*
Observe that $head(r_1) = head(r_2)$ and using $\sigma_1 = \{C \mapsto B, B \mapsto C\}$ and $\sigma_2 = \{B \mapsto C, C \mapsto B\}$, it follows that $r_1\sigma_1 = r_2$ and $r_2\sigma_2 = r_1$

The *body-variant* problem is to decide whether two rules differ only by the naming of body-only variables:

Definition 6 (**Body-variant problem**). Given rules r_1 and r_2 such that $head(r_1) = head(r_2)$, the *body-variant problem* is deciding whether r_1 and r_2 are body-variants of each other.

Deciding whether two rules are body variants is intractable as there is a reduction from the *graph isomorphism* problem to the body-variant problem, for which existing decision procedures are computationally prohibitive (Babai and Luks 1983; Babai and Codenotti 2008). The following result illustrates the (GI)-hardness (Köbler, Schöning, and Torán 1993) of the body-variant problem:

Proposition 1 (**Body-variant hardness**). The body-variant problem is GI-hard[1].

Proof (sketch). We encode a graph $G = (N, E)$ as a rule using a binary relation *edge/2* and $|N|$ variables denoting nodes of G. The reduction follows from this encoding (See Appendix). □

This result motivates us to develop an incomplete yet tractable approach to the body-variant problem. Before discussing our approach, we first generalise the body-variant problem to hypothesis variants:

Definition 7 (**Hypothesis variant**). A hypothesis h' is a *variant* of a hypothesis h if there is a bijective mapping f from the rules of h' to the rules of h such that for all $r \in h'$, r is a body variant $f(r)$.

Example 2. Consider the following hypotheses:

Hypothesis 1 (h_1)
r_1 = *zendo(A) ← piece(A,B), size(B,C), blue(B), small(C)*
r_2 = *zendo(A) ← piece(A,C), size(C,B), red(C), large(B)*

Hypothesis 2 (h_2)
r_3 = *zendo(A) ← piece(A,C), size(C,B), blue(C), small(B)*
r_4 = *zendo(A) ← piece(A,B), size(B,C), red(B), large(C)*

Using a mapping f such that $f(r_1) = r_3$ and $f(r_2) = r_4$, we can observe that h_1 is a hypothesis-variant of h_2.

We generalise the body-variant problem to hypotheses:

Definition 8 (**Hypothesis-variant problem**). Given hypotheses h_1 and h_2, the *hypothesis-variant problem* is deciding whether h_1 and h_2 are hypothesis-variants of each other.

The body-variant problem is a special case of the hypothesis-variant problem, where hypotheses contain a single rule. Thus, it follows that the hypothesis-variant problem is also GI-hard.

5 Tractable Symmetry Breaking for ILP

Due to the hardness of the hypothesis variant problem, we now describe a sound yet incomplete approach to remove hypothesis variants from an ILP hypothesis space. For simplicity, we describe the approach for a single-rule hypothesis before generalising to arbitrary hypotheses.

Informally, our approach exploits variable ordering in a rule and forces the following condition on all rules in the hypothesis space. If a body literal l does not contain a variable x, but contains variables both larger and smaller than x according to the variable ordering, then x must occur in a literal lexicographically smaller than l. In other words, any skipped occurrence of a variable, with respect to $<_{\mathcal{V}}$, must be justified by lexicographically smaller literals.

To illustrate the idea, reconsider this Zendo rule from the introduction:
r_2 = *zendo(A) ← piece(A,C), size(C,B), blue(C), small(B)*

[1]Arvind et al. (2015) reduce graph isomorphism to hypergraph isomorphism. Using this reduction we can extend our reduction from binary to n-ary predicates.

We order variables alphabetically, i.e. $A <_{\mathcal{V}} B <_{\mathcal{V}} C$. The variable B is not in *piece(A,C)* so we need to check if a lexicographically smaller variable tuple than (A, C) is in the rule. To compare tuples we first order the variables in the tuple and add a prefix containing occurrences of the smallest variable. We assume that rules do not contain singleton variables. Thus, the argument of a unary literal must appear elsewhere in the rule, i.e. how the literal is ordered is completely dependent on another literal. For instance, the tuple of *size(C,B)* is (B, C) and the tuple of *piece(A,C)* is (A, C). Observe that $(A, C) <_{lex} (B, C)$ and there are no tuples smaller than (A, C) derivable from the rule. Thus, r_2 is pruned because it does not contain a tuple smaller than (A, C) containing the skipped variable B. Observe that a similar analysis shows that r_1 from the introduction is not pruned.

To formalise our approach, we make several assumptions explicit:

(i) A rule contains only variables, no constants. For example, the rule *zendo(A) ← piece(A,B)* obeys this assumption, but *zendo(A) ← piece(A,lbt)* does not, where *lbt* is a constant.

(ii) The variables in the head of a rule are always the smallest variables according to the variable ordering[2]. Formally, for any rule r, $v_1 \in head_vars(r)$, $v_2 \in body_vars(r)$, $v_1 <_{\mathcal{V}} v_2$. For example, the rule *zendo(A) ← piece(A,B)* obeys this assumption as $A <_{\mathcal{V}} B$ while the rule *zendo(B) ← piece(A,B)* does not.

(iii) A variable is in a rule if and only if every smaller variable is in the rule. Formally, for any rule r, $v_1, v_2 \in vars(r)$, if there exists $v_3 \in \mathcal{V}$ such that $v_1 <_{\mathcal{V}} v_3 <_{\mathcal{V}} v_2$, then $v_3 \in vars(r)$. For example, the rule *zendo(A) ← piece(A,B)* obeys this assumption, while the rule *zendo(A) ← piece(A,C)* does not as $A <_{\mathcal{V}} B <_{\mathcal{V}} C$.

Any rule that violates assumptions (ii) and/or (iii) can be transformed by a variable renaming into the appropriate form, either by shifting variables or swapping head and body variables. Furthermore, we consider the argument tuples of literals reordered with respect to $<_{\mathcal{V}}$:

Definition 9 (**Ordered variables**). Let l be a literal, $args(l) = (x_1, \ldots, x_n)$, $1 \leq i_1, \cdots, i_n \leq n$ such that for all $1 \leq j < n$, $x_{i_j} \leq_{\mathcal{V}} x_{i_{j+1}}$ and $i_j \neq i_{j+1}$. Then $ord_vars(l) = (x_{i_1}, \ldots, x_{i_n})$.

Example 3. Let $A <_{\mathcal{V}} B <_{\mathcal{V}} D$, and $l = p(D, A, B)$ be a literal. Then $ord_vars(l) = (A, B, D)$.

Additionally, we add a prefix to the tuples resulting from Definition 9 to produce tuples of uniform length:

Definition 10 (**Prefix padding**). Let $k \geq 0$ and l be a literal such that $ord_vars(l) = (x_1, \ldots, x_n)$. Then

$$pre_pad_k(l) = (\underbrace{x_\star, \ldots, x_\star}_{\max\{0,k-n\} \text{ times}}, x_1, \ldots, x_n)$$

where $x_\star$ is the minimal element with respect to $<_{\mathcal{V}}$.

For the rest of this section we assume the following:

[2]The total order $<_{\mathcal{V}}$ on variable symbols.

Assumption 1. Let p be a predicate symbol in the BK such that for all other predicate symbols q in the *BK*, $arity(p) \geq arity(q)$. Then $k \geq arity(p)$.

We define a lexicographical literal order to order literals by first rearranging their arguments with respect to $<_{\mathcal{V}}$ and then adding a prefix to the resulting tuples to produce tuples of uniform size:

Definition 11 (**Lexicographical literal order**). Let l_1 and l_2 be literals with arity ≥ 2. Then we say that $l_1 <^k_{lex} l_2$ if $pre_pad_k(l_1) <_{lex} pre_pad_k(l_2)$ where $<_{lex}$ is the lexicographical order on k-tuples.

Example 4. Let $l_1 = p(D, A, B)$ and $l_2 = q(C, B)$ be literals where variables are alphabetically ordered and $k = 3$. Then $l_2 <^3_{lex} l_1$ because $ord_vars(l_1) = (A, B, D)$, $ord_vars(l_2) = (B, C)$, $pre_pad_3(l_1) = (A, B, D)$, $pre_pad_3(l_1) = (x_*, B, C)$, and $(x_*, B, C) <_{lex} (A, B, D)$ where $x_* = A$.

We use this order to identify literals with *skipped* variables:

Definition 12 (**Skipped**). Let l be a literal such that $pre_pad_k(l) = (x_1, \ldots, x_n)$. Then $skipped_k(l) = \{y \mid x_1 <_{\mathcal{V}} y <_{\mathcal{V}} x_n \wedge y \notin vars(l)\}$.

Example 5. Consider the rule $h(A, B) \leftarrow p(A, E), p(B, C), p(C, D)$ where variables are ordered alphabetically. Observe that $skipped_2(p(A, E)) = \{B, C, D\}$ and $skipped_2(p(C, D)) = \emptyset$.

Rules can contain literals with skipped variables but the skipped variables must be witnessed by a literal lower in the lexicographical literal order:

Definition 13 (**Witnessed**). Let r be a rule, v be a variable, $l_1 \in body_{\geq 2}(r)$ such that $v \in skipped_k(l_1)$, and $l_2 \in body_{\geq 2}(r)$ such that $v \in vars(l_2)$ and $l_2 <^k_{lex} l_1$. Then we say that l_1 is v-witnessed in r.

A variable is *unsafe* if it is skipped in a literal and there is no witnessing literal:

Definition 14 (**Safe variable**). Let r be a rule and $v \in body_vars(r)$ such that for all $l \in body_{\geq 2}(r)$, where $v \in skipped_k(l)$, l is v-witnessed in r. Then v is *safe*. Otherwise, v is *unsafe*.

We illustrate the concept of safe variables:

Example 6. Consider the rules:
r_2 = *h(A,B) ← p(A,C),p(B,E),p(C,D)*
r_3 = *h(A,B) ← p(A,C),p(B,D),p(C,E)*
Observe that $p(B, E)$ skips D and none of the literals of r_2 witness $p(B, E)$. The only literal containing D is $p(C, D)$ and $p(B, E) <^2_{lex} p(C, D)$. Thus, D is *unsafe* in r_2. In rule r_3, $p(C, E)$ skips D and $p(C, E)$ is witnessed by $p(B, D)$ because $p(B, D) <^2_{lex} p(C, E)$. Thus, D is safe in r_3. Observe that in r_3 all variables are safe including C and E, i.e. $p(A, C) <^2_{lex} p(B, D)$.

We now show that every rule has a body-variant containing only safe variables:

Proposition 2 (**Soundness**). For every rule r there exists a rule r' such that r' is a body-variant of r and all variables in r' are safe.

Proof (sketch). To simplify our argument we assume that $vars(r) = \{x_1, \cdots, x_m\}$ where for $1 \leq j < m$, $x_j <_{\mathcal{V}} x_{j+1}$. We prove the proposition by induction,[3] selecting the smallest unsafe variable x (according to $<_{\mathcal{V}}$) and then constructing a substitution that, when applied to r, results in a rule where the smallest unsafe variable is larger than x. After finitely many steps the smallest unsafe variable is larger than x_m, i.e. every variable in the constructed rule is safe. □

See the appendix for a full proof. Below we outline the transformation used by the induction step of the full proof.

Example 7. Let

$$r_8 = h(A, B) \leftarrow p(A, E), p(B, \mathbf{C}), p(C, D).$$

where the variables are ordered alphabetically. The argument outlined in the proof of Proposition 2 applies to rule r_8 as follows. Observe that C is the smallest unsafe variable in r_8. Applying $\sigma_1 = \{E \mapsto C, C \mapsto F\}\{F \mapsto E\}$ to r_8 we get the rule:

$$r_9 = h(A, B) \leftarrow p(A, C), p(B, E), p(E, \mathbf{D}).$$

Now D is the smallest unsafe variable in r_9. Applying $\sigma_2 = \{E \mapsto D, D \mapsto F\}\{F \mapsto E\}$ to r_9 we get the rule:

$$r_{10} = h(A, B) \leftarrow p(A, C), p(B, D), p(D, E).$$

Observe that rule r_{10} does not have unsafe variables. Thus applying the substitution $\sigma = \sigma_1\sigma_2 = \{E \mapsto C, C \mapsto D, D \mapsto E\}$ to r results in a body-variant where all variables are safe.

We now show our main result:

Theorem 1. Let $(E, B, \mathcal{H})$ be an ILP input, $cost_{E,B}$ be a cost function, and $h \in \mathcal{H}$ be an optimal hypothesis with respect to $cost_{E,B}$. Then there exists $h' \in \mathcal{H}$ such that h' is a variant of h and all rules of h' only contain safe variables.

See the appendix for examples of rules that are body-variants of each other and only contain safe variables.

6 Implementation

We demonstrate our idea in ASP with the ILP system POPPER (Cropper and Morel 2021; Cropper and Hocquette 2023a).

Popper

Our symmetry-breaking approach directly works with all variants of POPPERhopper,maxsynth,propper. For simplicity, we describe the basic version of POPPER.

POPPER uses a generate, test, combine, and constrain loop to find an optimal hypothesis (Definition 3) (the full algorithm is in the appendix). POPPER starts with an answer set program $\mathcal{P}$. This program can be viewed as a *generator* program because each model (answer set) represents a hypothesis. The program $\mathcal{P}$ uses head (*hlit*/3) and body (*blit/3*) literals to represent a hypothesis. The first argument of each literal is the rule ID, the second is the predicate symbol, and the third is the literal variables, where *0* represents *A*, *1* represents *B*, etc. For instance, POPPER represents the rule *f(A,B)* ← *tail(A,C), head(C,B)* as a set with three atoms:

```
{hlit(0,f,(0,1)), blit(0,tail,(0,2)),
 blit(0,head,(2,1))}
```

The program $\mathcal{P}$ contains choice rules for head and body literals:

```
{hlit(Rule,Pred,Vars)}:-
  rule(Rule), vars(Vars,Arity),  hpred(Pred,Arity).
{blit(Rule,Pred,Vars)}:-
  rule(Rule), vars(Vars,Arity), bpred(Pred,Arity).
```

The literal *rule(Rule)* denotes rule indices. The literals *hpred(Pred,Arity)* and *bpred(Pred,Arity)* denote predicate symbols and arities that may appear in the head or body of a rule, respectively. The literal *vars(Vars,Arity)* denotes all possible variable tuples.

In the *generate stage*, POPPER uses an ASP system to find a model of $\mathcal{P}$ for a fixed hypothesis size, enforced via a cardinality constraint on the number of head and body literals. If no model is found, POPPER increments the hypothesis size and loops again. If a model exists, POPPER converts it to a hypothesis h (a definite program).

In the *test stage*, POPPER uses Prolog to test h on the training examples and background knowledge. If h entails at least one positive example and no negative examples, POPPER saves h as a *promising program*.

In the *combine stage*, POPPER searches for a combination (a union) of promising programs that entails all the positive examples and has minimal size. POPPER formulates the search as a combinatorial optimisation problem (Cropper and Hocquette 2023a), implemented in ASP as an optimisation problem. If a combination exists, POPPER saves it as the best hypothesis and updates the maximum hypothesis size.

In the *constrain stage*, POPPER uses h to build constraints, which it adds to $\mathcal{P}$ to prune models and thus prune the hypothesis space. For instance, if h does not entail any positive example, POPPER adds a constraint to prune its specialisations as they are guaranteed not to entail any positive example. For instance, the following constraint prunes all specialisations (supersets) of the rule *f(A,B)* ← *tail(A,C), head(C,B)*:

```
:- hlit(R,f,(0,1)), blit(R,tail,(0,2)),
   blit(R,head,(2,1)).
```

POPPER repeats this loop using multi-shot solving (Gebser et al. 2019) and terminates when it exhausts the models of $\mathcal{P}$ or exceeds a user-defined timeout. It then returns the best hypothesis found.

Symmetry Breaking Encoding

We now describe our ASP encoding $\mathcal{E}$ to break symmetries. We add $\mathcal{E}$ to the answer set program $\mathcal{P}$ used by POPPER to generate programs to prune unsafe rules and thus unsafe hypotheses.

The fact `var_member(Vars,V)` denotes that variable `V` is a member of variable tuple `Vars`, e.g. `var_member((0,4,3),3)`. For every variable tuple `xs`, we sort the tuple to `ys` and add the fact `ordered_vars(xs,ys)` to $\mathcal{E}$. For instance, for the variable tuple `(4,1,3)`, we add the fact `ordered_vars((4,1,3),(1,3,4))`. These facts match *ordered variables* (Definition 9). We add facts to encode the lexicographic order over variable tuples (Definition 11). For instance, we add the facts `lower((0,0,1),(0,0,2))` and `lower((4,7,1),(4,8,2))`. We add all skipped facts

[3]Induction is performed over a finite total ordering towards a maximum element.

of the form `skipped(Vars,V)` to denote that the variable `V` is strictly between two variables `A` and `B` in an ordered variable tuple `Vars`, where `V` is not in `Vars` (Definition 12); these include, for instance, the facts `skipped((0,1,3),2)`, `skipped((1,3,5),2)`, and `skipped((1,3,5),4)`. We add a rule to identify the ordered variable tuple of a selected body literal:

```
appears(Rule,OrderedVars):-
  blit(Rule,_,Vars), padded_vars(Vars,PaddedVars),
  ordered_vars(PaddedVars,OrderedVars).
```

The literal `padded_vars(Vars,PaddedVars)` performs prefix padding on variable tuples (Definition 10). For instance, assuming that the maximum arity of any literal is 4, for the variable tuple `(4,1)` we added the fact `padded_vars((4,1),(0,0,4,1))`. We add a rule to identify witnessed variables (Definition 13):

```
witnessed(Rule,V,Vars1):-
   appears(Rule,Vars1), skipped(Vars1,V),
   lower(Vars2,Vars1), var_member(V,Vars2),
   appears(Rule,Vars2).
```

Finally, we add a constraint to prune rules with unsafe variables 14):

```
:- body_var(Rule,V), appears(Rule,Vars),
   skipped(Vars,V), not witnessed(Rule,V,Vars).
```

Our symmetry-breaking encoding adds at most $O(m \cdot n^2 \cdot k)$ ground rules to the ASP solver, where m is the number of candidate rules in a hypothesis, n is the number of possible variable tuples, and k is the number of possible variables.

7 Experiments

To test our claim that pruning variants can reduce solving time, our experiments aim to answer the question:

Q1 Can symmetry breaking reduce solving time?

To answer **Q1**, we compare the solving time of POPPER with and without symmetry breaking.

To test our claim that pruning variants allows us to scale to harder tasks, our experiments aim to answer the question:

Q2 Can symmetry breaking reduce solving time when progressively increasing the complexity of tasks?

To answer **Q2**, we compare the solving time of POPPER with and without symmetry breaking when varying the complexity of an ILP task.

Finally, the goal of breaking symmetries is to reduce solving time and, in turn, reduce learning time. Therefore, our experiments aim to answer the question:

Q3 Can symmetry breaking reduce learning time?

To answer **Q3**, we compare the learning time of POPPER with and without symmetry breaking.

Experimental Setup To answer **Q1**, we compare the solving time (time spent generating hypotheses) of POPPER with and without symmetry breaking. We use a solving timeout of 20 minutes per task. To answer **Q2**, we compare the solving time of POPPER with and without symmetry breaking on progressively harder ILP tasks by increasing the number of variables allowed in a rule[4]. We use a solving timeout of 60 minutes per task. For **Q1** and **Q2**, the independent variable is whether POPPER uses symmetry breaking and the dependent variable is the solving time, which depends entirely on the independent variable. To answer **Q3**, we compare the learning time of POPPER with and without symmetry breaking. We use a learning timeout of 60 minutes per task. However, learning times are a function of many things, not only solving time. For instance, symmetry breaking could reduce solving time by 50% allowing POPPER to generate twice as many hypotheses but it also needs to test them. Therefore, learning time does not directly measure the impact of symmetry breaking. In other words, the dependent variable (learning time) does not entirely depend on the independent variable.

In all experiments, the runtimes include the time spent generating the symmetry-breaking rules. We round times over one second to the nearest second. We repeat all experiments 10 times. We plot and report 95% confidence intervals (CI). We compute 95% CI via bootstrapping when data is non-normal. To determine statistical significance, we apply either a paired t-test or the Wilcoxon signed-rank test, depending on whether the differences are normally distributed. We use the Benjamini–Hochberg procedure to correct for multiple comparisons. We use POPPER 4.4.0. We use an AWS m6a.16xlarge instance to run experiments where each learning task uses a single core.

Domains We use 449 learning tasks from several domains:

1D-ARC. This dataset (Xu et al. 2024) contains visual reasoning tasks inspired by the abstract reasoning corpus (Chollet 2019).

IGGP. The task is to induce rules from game traces (Cropper, Evans, and Law 2020) from the general game playing competition (Genesereth and Björnsson 2013).

IMDB. A real-world dataset which contains relations about movies (Mihalkova, Huynh, and Mooney 2007).

List functions. The goal of each task in this data (Rule et al. 2024) is to identify a function that maps input lists to output lists, where list elements are natural numbers.

Trains. The goal is to find a hypothesis that distinguishes east and west trains (Larson and Michalski 1977).

Zendo. An inductive game where players discover secret rules by building structures (Cropper and Hocquette 2023b).

Experimental Results

Q1. Can Symmetry Breaking Reduce Solving Time? Figure 2 shows the solving times of POPPER with and without symmetry breaking[5]. Significance tests confirm $(p < 0.05)$ that symmetry breaking reduces solving times on 97/449 (22%) tasks and increases solving times on 1/449 (0%) tasks. There is no significant difference in the other tasks. The mean decrease in solving time is 178 ± 56 seconds. The median decrease is 20 seconds with 95% CI between 8 and 51 seconds. The mean and median increase is 41 seconds. Some

[4]Cropper and Morel (2021) show that the hypothesis space grows exponentially in the number of variables allowed in a rule.

[5]Detailed per-task results and corresponding improvements are in the appendix.

improvements are substantial. For instance, for the *sokoban-terminal* task, symmetry breaking reduces the solving time from 1075 ± 112 seconds to 59 ± 6 seconds. These are minimum improvements because POPPER without symmetry breaking often times out after 20 minutes. With a longer timeout, we would likely see greater improvements. Overall, the results show that symmetry breaking can drastically reduce solving time.

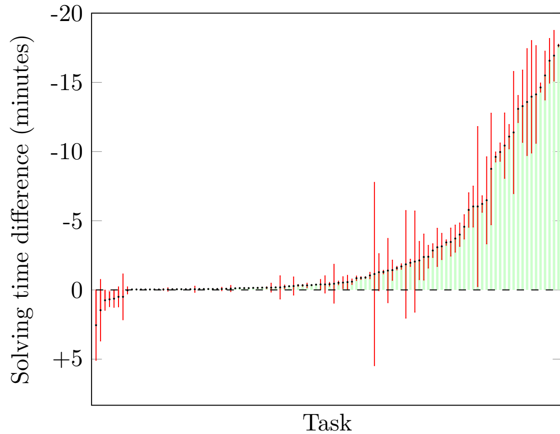

Figure 2: Solving time difference (minutes) with symmetry breaking. The tasks are ordered by improvement.

Q2. Can Symmetry Breaking Reduce Solving Time When Progressively Increasing the Complexity of Tasks? Figure 3 shows the solving times of POPPER with and without symmetry breaking on progressively harder ILP tasks. The results show that POPPER without symmetry breaking struggles to scale to harder tasks. Without symmetry breaking, the mean solving times when allowed to use 7, 8, or 9 variables in a rule are 125 ± 44, 1411 ± 310, and 3600 ± 0 seconds respectively. By contrast, with symmetry breaking, the mean solving times when allowed to use 7, 8, or 9 variables in a rule are 4 ± 0, 7 ± 0, and 17 ± 2 seconds respectively. In other words, for the hardest task, symmetry breaking reduces solving time from over an hour to only 17 seconds, a 99.5% reduction. Overall, the results show that symmetry breaking can reduce solving time when progressively increasing the complexity of tasks.

Q3. Can Symmetry Breaking Reduce Learning Time? Figure 4 shows the learning times of POPPER with and without symmetry breaking. Significance tests confirm ($p < 0.05$) that symmetry breaking reduces learning times on 128/449 (29%) tasks and increases learning times on 12/449 (3%) tasks. There is no significant difference in the other tasks. The mean decrease in learning time is 385 ± 109 seconds and the median is 54 seconds with 95% CI between 21 and 134 seconds. The mean increase is 192 ± 121 seconds and the median is 120 seconds with 95% CI between 54 and 266 seconds. These are minimum improvements because POPPER without symmetry breaking often times out after 60 minutes. With a longer timeout, we would likely see greater improvements. Overall, the results show that symmetry breaking can

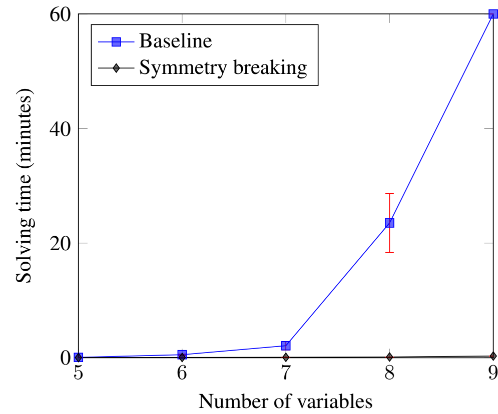

Figure 3: Solving times (minutes) of POPPER with and without (baseline) symmetry breaking on one *trains* task. We vary the number of variables allowed in a rule and thus the size of the hypothesis space.

drastically reduce learning time.

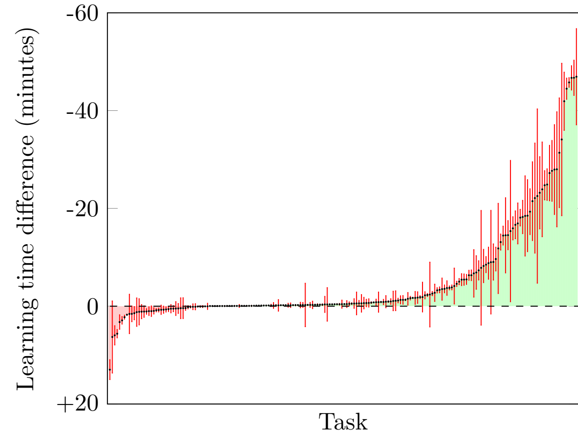

Figure 4: Learning time difference (minutes) with symmetry breaking. The tasks are ordered by improvement.

8 Conclusions and Limitations

We introduced a symmetry-breaking method for ILP. We showed that determining whether two rules are body-variants (Definition 6) is graph isomorphism hard (Proposition 1). We described a symmetry-breaking approach to prune body-variant rules and implemented it in ASP. We proved that this approach is sound and only prunes body-variant rules (Proposition 2). We have experimentally shown on multiple domains, including visual reasoning and game playing, that our approach can drastically reduce solving time and learning time, sometimes from over an hour to only 17 seconds.

Limitations Our symmetry-breaking approach is incomplete, as complete symmetry-breaking is graph isomorphism hard (Proposition 1). Finding other efficient symmetry-breaking techniques is future work.

Acknowledgments

We thank Andreas Niskanen for early discussions about this work. Andrew Cropper was supported by his EPSRC fellowship (EP/V040340/1). David M. Cerna was supported by the Czech Science Foundation Grant 22-06414L and Cost Action CA20111 EuroProofNet. Matti Jarvisalo was supported by Research Council of Finland (grant 356046).

References

Ahlgren, J.; and Yuen, S. Y. 2013. Efficient program synthesis using constraint satisfaction in inductive logic programming. *J. Machine Learning Res.*, 14(1): 3649–3682.

Aloul, F. A.; Markov, I. L.; and Sakallah, K. A. 2003. Shatter: efficient symmetry-breaking for boolean satisfiability. In *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, 836–839. ACM.

Anders, M.; Brenner, S.; and Rattan, G. 2024. Satsuma: Structure-Based Symmetry Breaking in SAT. In Chakraborty, S.; and Jiang, J. R., eds., *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*, volume 305 of *LIPIcs*, 4:1–4:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Arvind, V.; Das, B.; Köbler, J.; and Toda, S. 2015. Colored Hypergraph Isomorphism is Fixed Parameter Tractable. *Algorithmica*, 71(1): 120–138.

Babai, L.; and Codenotti, P. 2008. Isomorhism of Hypergraphs of Low Rank in Moderately Exponential Time. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, 667–676.

Babai, L.; and Luks, E. M. 1983. Canonical labeling of graphs. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, 171–183. New York, NY, USA: Association for Computing Machinery. ISBN 0897910990.

Bembenek, A.; Greenberg, M.; and Chong, S. 2023. From SMT to ASP: Solver-Based Approaches to Solving Datalog Synthesis-as-Rule-Selection Problems. *Proc. ACM Program. Lang.*, 7(POPL).

Blockeel, H.; and De Raedt, L. 1998. Top-Down Induction of First-Order Logical Decision Trees. *Artif. Intell.*, 101(1-2): 285–297.

Chollet, F. 2019. On the Measure of Intelligence. *CoRR*.

Corapi, D.; Russo, A.; and Lupu, E. 2011. Inductive Logic Programming in Answer Set Programming. In *ILP 2011*.

Crawford, J. M.; Ginsberg, M. L.; Luks, E. M.; and Roy, A. 1996. Symmetry-Breaking Predicates for Search Problems. In Aiello, L. C.; Doyle, J.; and Shapiro, S. C., eds., *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996*, 148–159. Morgan Kaufmann.

Cropper, A.; and Dumancic, S. 2022. Inductive Logic Programming At 30: A New Introduction. *J. Artif. Intell. Res.*, 74: 765–850.

Cropper, A.; Evans, R.; and Law, M. 2020. Inductive general game playing. *Mach. Learn.*, 109(7): 1393–1434.

Cropper, A.; and Hocquette, C. 2023a. Learning Logic Programs by Combining Programs. In *ECAI 2023 - 26th European Conference on Artificial Intelligence*, volume 372, 501–508. IOS Press.

Cropper, A.; and Hocquette, C. 2023b. Learning Logic Programs by Discovering Where Not to Search. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023*, 6289–6296. AAAI Press.

Cropper, A.; and Morel, R. 2021. Learning programs by learning from failures. *Mach. Learn.*, 110(4): 801–856.

Darga, P. T.; Liffiton, M. H.; Sakallah, K. A.; and Markov, I. L. 2004. Exploiting structure in symmetry detection for CNF. In Malik, S.; Fix, L.; and Kahng, A. B., eds., *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, 530–534. ACM.

De Raedt, L. 2008. *Logical and relational learning*. ISBN 978-3-540-20040-6.

De Raedt, L.; and Dehaspe, L. 1997. Clausal Discovery. *Mach. Learn.*, 26(2-3): 99–146.

Devriendt, J.; and Bogaerts, B. 2016. BreakID: Static Symmetry Breaking for ASP (System Description). *CoRR*, abs/1608.08447.

Devriendt, J.; Bogaerts, B.; Bruynooghe, M.; and Denecker, M. 2016. On local domain symmetry for model expansion. *Theory Pract. Log. Program.*, 16(5-6): 636–652.

Drescher, C.; Tifrea, O.; and Walsh, T. 2011. Symmetry-breaking answer set solving. *AI Commun.*, 24(2): 177–194.

Evans, R.; Hernández-Orallo, J.; Welbl, J.; Kohli, P.; and Sergot, M. J. 2021. Making sense of sensory input. *Artif. Intell.*, 293: 103438.

Fürnkranz, J.; and Kliegr, T. 2015. A Brief Overview of Rule Learning. In *RuleML 2015*, volume 9202 of *Lecture Notes in Computer Science*, 54–69. Springer.

Galárraga, L.; Teflioudi, C.; Hose, K.; and Suchanek, F. M. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6): 707–730.

Garey, M. R.; and Johnson, D. S. 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1): 27–82.

Genesereth, M. R.; and Björnsson, Y. 2013. The International General Game Playing Competition. *AI Magazine*, 34(2): 107–111.

Hocquette, C.; Niskanen, A.; Järvisalo, M.; and Cropper, A. 2024. Learning MDL Logic Programs from Noisy Data. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI*, 10553–10561. AAAI Press.

Junttila, T. A.; and Kaski, P. 2007. Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs. In *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, Louisiana, USA, January 6, 2007*. SIAM.

Kaminski, T.; Eiter, T.; and Inoue, K. 2019. Meta-Interpretive Learning Using HEX-Programs. In *IJCAI 2019*, 6186–6190.

Köbler, J.; Schöning, U.; and Torán, J. 1993. *The Graph Isomorphism Problem: Its Structural Complexity*. Progress in Theoretical Computer Science. Birkhäuser/Springer. ISBN 978-1-4612-6712-6.

Larson, J.; and Michalski, R. S. 1977. Inductive inference of VL decision rules. *SIGART Newsletter*, 63: 38–44.

Law, M.; Russo, A.; and Broda, K. 2014. Inductive Learning of Answer Set Programs. In *JELIA 2014*.

Lloyd, J. W. 2012. *Foundations of logic programming*. Springer Science & Business Media.

McKay, B. D. 1981. Practical Graph Isomorphism. *Congressus Numerantium*, 30: 45–87.

Mihalkova, L.; Huynh, T. N.; and Mooney, R. J. 2007. Mapping and Revising Markov Logic Networks for Transfer Learning. In *AAAI 2007*, 608–614.

Muggleton, S. 1991. Inductive Logic Programming. *New Generation Computing*, 8(4): 295–318.

Muggleton, S. 1995. Inverse Entailment and Progol. *New Generation Comput.*, 13(3&4): 245–286.

Plotkin, G. 1971. *Automatic Methods of Inductive Inference*. Ph.D. thesis, Edinburgh University.

Quinlan, J. R. 1990. Learning Logical Definitions from Relations. *Mach. Learn.*, 5: 239–266.

Raedt, L. D.; and Ramon, J. 2004. Condensed Representations for Inductive Logic Programming. In Dubois, D.; Welty, C. A.; and Williams, M., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, 438–446. AAAI Press.

Rule, J. S.; Piantadosi, S. T.; Cropper, A.; Ellis, K.; Nye, M.; and Tenenbaum, J. B. 2024. Symbolic metaprogram search improves learning efficiency and explains rule learning in humans. *Nature Communications*, 15(1): 6847.

Schüller, P.; and Benz, M. 2018. Best-effort inductive logic programming via fine-grained cost-based hypothesis generation - The inspire system at the inductive logic programming competition. *Mach. Learn.*, 107(7): 1141–1169.

Shapiro, E. Y. 1983. *Algorithmic Program DeBugging*. Cambridge, MA, USA. ISBN 0262192187.

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

Struyf, J.; and Blockeel, H. 2003. Query Optimization in Inductive Logic Programming by Reordering Literals. In Horváth, T., ed., *Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings*, volume 2835 of *Lecture Notes in Computer Science*, 329–346. Springer.

Tamaddoni-Nezhad, A.; and Muggleton, S. H. 2009. The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. *Mach. Learn.*, 76(1): 37–72.

Tarzariol, A.; Gebser, M.; and Schekotihin, K. 2021. Lifting Symmetry Breaking Constraints with Inductive Logic Programming. In Zhou, Z., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021*, 2062–2068. ijcai.org.

Xu, Y.; Li, W.; Vaezipoor, P.; Sanner, S.; and Khalil, E. B. 2024. LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations. *Trans. Mach. Learn. Res.*, 2024.

Zeman, V.; Kliegr, T.; and Svátek, V. 2021. RDFRules: Making RDF rule mining easier and even more efficient. *Semantic Web*, 12(4): 569–602.

Zeng, Q.; Patel, J. M.; and Page, D. 2014. QuickFOIL: Scalable Inductive Logic Programming. *VLDB*, 197–208.