Hypothesis Space Processing for Efficient Rule Learning Through Inductive Logic Programming

David M. Cerna*

Dynatrace Research, Linz, Austria

May 13, 2025

Inductive Logic programming (ILP) combines knowledge representation and machine learning [7, 3]. The goal is to search a hypothesis space, consisting of logic programs, to find a hypothesis that generalizes the given training examples and background knowledge. Early ILP approaches such as *foil* iteratively specialize rules till only positive examples are accepted and repeat this process until all positive examples are accepted. This process was refined in systems like *Aleph* by restricting the search space of specializations to hypotheses that θ-subsume the so-called *bottom clause*, i.e, the most specific clause which accepts a given positive example.

Modern ILP often follows the *meta-learning* approach, i.e., perform search through an encoding of the hypothesis space. In particular, *Popper* [4] encodes the generation of logic programs modulo the background knowledge as an ASP program (*generate phase*). As *Popper* tests the hypothesis produced by the generate phase, additional constraints are added guiding the search at subsequent steps. This leads to a significant decrease in the number of tested hypotheses while maintaining optimality guarantees. These constraints are based on propositional subsumption and are thus computationally feasible to check. Popper's approach to ILP has led to significant advancements in the tasks such systems can handle. For example learning with *negative invention* [1], *Higher-order* [8], *very large programs* [5], and competative performance on ARC [6].

In this abstract, we focus on alternative constraints based on the semantics of the background knowledge and **symmetry breaking**, which can improve performance in all of the above areas. For example, in [2] we introduce the notion of *reducible* literal, i.e., given a rule r containing l, the following holds: $B \models r \leftrightarrow r \setminus \{l\}$ where B is the background knowledge. We use this concept to introduce additional constraints while learning. In some cases, we can constrain the hypothesis space prior to learning, for example, if a predicate in the background knowledge is satisfied by a finite number of instantiations. This limits the number of times it can occur in a rule. For example, consider the following where $r_1 \preceq_\theta r_2$:

*This extended abstract concerns joint work with Andrew Cropper.

B = { *edge(a,b), edge(b,c), edge(c,a)* }
r_1 = $h \leftarrow$ *edge(A,B), edge(B,C), edge(C,D), edge(D,E)*
r_2 = $h \leftarrow$ *edge(A,B), edge(B,C), edge(C,A)*
$\theta = \{D \mapsto A, E \mapsto B\}$

We refer to this property as *recall redundant*, similar properties are definable for literal combinations that make a rule unsat, literal combinations that are always true, and implication relations between literals. A final approach to constraining the hypothesis space is to remove rule variants. This is achieved by ordering the literals by their argument tuples and normalizing with respect to certain properties.

$$r_1 : h(A, B) \leftarrow p(A, E), p(B, \mathbf{C}), p(C, D).$$

$$r_2 : h(A, B) \leftarrow p(A, C), p(B, E), p(C, \mathbf{D}).$$

Observe $r_1\sigma_2 = r_2$ where $\sigma_2 = \{E \mapsto C, C \mapsto E\}\{E \mapsto D\}$. In the above case, we order the variables and ensure that variables are witnessed by smaller variables (with respect to a variable order). This normalization is sound (does not remove all variants), but incomplete. We will cover these approaches in more detail during the talk.

References

[1] David M. Cerna and Andrew Cropper. Generalisation through negation and predicate invention. *AAAI*, 38(9):10467–10475, 2024.

[2] Andrew Cropper and David M. Cerna. Efficient rule induction by ignoring pointless rules, 2025.

[3] Andrew Cropper and Sebastijan Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022.

[4] Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Mach. Learn.*, 110(4):801–856, 2021.

[5] Céline Hocquette, Andreas Niskanen, Rolf Morel, Matti Järvisalo, and Andrew Cropper. Learning big logical rules by joining small rules. In *IJCAI*, pages 3430–3438, 2024.

[6] Céline Hocquette and Andrew Cropper. Relational decomposition for program synthesis, 2025.

[7] Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[8] Stanislaw J. Purgal, David M. Cerna, and Cezary Kaliszyk. Learning higher-order logic programs from failures. In Luc De Raedt, editor, *IJCAI*, pages 2726–2733, 2022.