# Can predicate invention compensate for incomplete background knowledge?

Andrew CROPPER Stephen MUGGLTEON
*Imperial College London, United Kingdom*

**Abstract.** In machine learning we are often faced with the problem of incomplete data, which can lead to lower predictive accuracies in both feature-based and relational machine learning. It is therefore important to develop techniques to compensate for incomplete data. In inductive logic programming (ILP) incomplete data can be in the form of missing values or missing predicates. In this paper, we investigate whether an ILP learner can compensate for missing background predicates through predicate invention. We conduct experiments on two datasets in which we progressively remove predicates from the background knowledge whilst measuring the predictive accuracy of three ILP learners with differing levels of predicate invention. The experimental results show that as the number of background predicates decreases, an ILP learner which performs predicate invention has higher predictive accuracies than the learners which do not perform predicate invention, suggesting that predicate invention can compensate for incomplete background knowledge.

**Keywords.** inductive logic programming, predicate invention, relational machine learning, meta-interpretive learning

## 1. Introduction

In an ideal world machine learning datasets would be complete. In reality, however, we are often faced with incomplete data, which can lead to lower predictive accuracies in both feature-based [6,10] and relational [22,17] machine learning. It is therefore important to develop techniques to compensate for incomplete data.

In inductive logic programming (ILP) [11], a form of relational machine learning, incomplete background knowledge can be in the form of missing values or missing predicates [9]. There are several techniques in the literature for handling missing data in an ILP setting [4,9]. In this paper, we investigate whether predicate invention [23,16] can compensate for incomplete data by inventing new predicates to replace those missing. We conduct experiments on two datasets in which we progressively remove predicates from the background knowledge whilst measuring the predictive accuracy of three ILP systems: Metagol$_{pi}$ [15], Metagol$_{nopi}$, and Progol [12], which can, cannot, and cannot perform predicate invention respectively (all described in Section 3). To our knowledge, this is the first study to investigate whether predicate invention can compensate for incomplete background knowledge. The main results of this paper are as follows:

- The experimental results show that a learner which performs predicate invention (Metagol$_{pi}$) outperforms learners which do not support predicate invention when

supplied with missing background predicates, indicating the predicate invention can compensate for incomplete background knowledge.
- In some cases, a learner which performs predicate invention (Metagol$_{pi}$) maintains *respectable* predictive accuracies with only half of the original background predicates, i.e. predicative accuracies do not decrease in proportion to the decrease in background predicates.

The rest of the paper is organised as follows. Section 2 gives an overview of predicate invention and meta-interpretive learning (MIL) [14,15], an ILP learning framework which supports predicate invention and upon which Metagol$_{pi}$ is based. Section 3 details the experiments on two kinship datasets. Section 4 discusses the results of the experiments and provides additional examples, including where predicate invention recovers missing predicates when learning robot plans. Finally, Section 5 concludes the paper.

## 2. Learning setting

This section describes predicate invention and provides an overview of MIL. We start by introducing the relevant terms and concepts from logic programming and ILP.

### *2.1. Logical notation*

A variable is denoted as alphanumerics beginning with upper-case letters. A constant is denoted as alphanumerics beginning with lower-case letters. A predicate symbol is denoted as alphanumerics starting with lower-case letters with an associated adicity, where the adicity of a predicate symbol is the number of arguments it takes. Predicate symbols are monadic when they have adicity one and dyadic when they have adicity two. Variables and constants are terms. A variable is first-order if it can be substituted for a term. A variable is higher-order if it can be substituted for a predicate symbol. A predicate symbol or higher-order variable immediately followed by a bracketed n-tuple of terms is an atom. The negation symbol is $\neg$. Both A and $\neg$A are literals whenever A is an atom, where A is a positive literal and $\neg$A is a negative literal. A finite (possibly empty) set of literals is a clause. A clause represents the disjunction of its literals. The clause $\{A_1,\ldots,A_m,\neg B_1,\ldots,\neg B_n\}$ is normally represented in logic programming as $A_1,\ldots,A_m \leftarrow B_1,\ldots,B_n$ where $\leftarrow$ is the logical implication connective. A Horn clause is a clause with at most one positive literal. A unit clause is a Horn clause with exactly one literal. A positive unit clause (sometimes called a fact) is unit clause with a positive literals. A negative unit clause is a unit clause with a negative literal. A denial or goal is a Horn clause which contains no positive literals. A definite clause is a clause that contains exactly one positive literal. A definite program is a finite set of definite clauses.

A standard ILP problem consists of an input $\langle B,E\rangle$ and an output $H$, where $B$ is a definite program representing the background knowledge, $E = \langle E^+,E^-\rangle$ are positive and negative examples respectively, and $H$ is definite program hypothesis. The goal of the ILP learner is to find a hypothesis $H$ such that $B,H \models E^+$ and $B,H \not\models E^-$. The following example illustrates the ILP problem setting for learning the grandparent kinship relation.

**Example 2.1.**

$$B = \begin{cases} mother(amy, amelia). \\ mother(ann, amy). \\ mother(linda, gavin). \\ father(gavin, amelia). \\ father(steve, amy). \end{cases}$$

$$E^+ = \begin{cases} grandparent(ann, amelia). \\ grandparent(steve, amelia). \\ grandparent(linda, amelia). \end{cases}$$

$$E^- = \begin{cases} grandparent(amy, amelia). \end{cases}$$

For the aforementioned grandparent problem, the following hypothesis is consistent with the examples:

$$grandparent(X,Y) \leftarrow father(X,Z), father(Z,Y)$$
$$grandparent(X,Y) \leftarrow father(X,Z), mother(Z,Y)$$
$$grandparent(X,Y) \leftarrow mother(X,Z), mother(Z,Y)$$
$$grandparent(X,Y) \leftarrow mother(X,Z), father(Z,Y)$$

### *2.2. Predicate invention*

We now describe how predicate invention can be used to improve the performance of an ILP learner.

**Definition 2.1.** The non-empty set of all predicate symbols (with their adicities) in a logic program is named the *predicate signature*.

**Example 2.2.** The predicate signature for Example 2.1 is as follows:

$$\{grandparent/2, mother/2, father/2\}$$

Predicate invention is the automatic introduction of predicate symbols into the language, i.e. the addition of predicate symbols into the predicate signature previously unseen in the examples and background knowledge [18]. There are two primary reasons for introducing a predicate symbol into the language: program reformulation and bias shift [23].

The idea behind program reformulation is to restructure a program to make it more readable. For example, for grandparent kinship relation in Example 2.1, learner can introduce a predicate symbol *p1/2* representing the parent kinship relation to learn a more compact hypothesis of the following form:

$$grandparent(X,Y) \leftarrow p1(X,Z), p1(Z,Y)$$
$$p1(X,Y) \leftarrow father(X,Y)$$
$$p1(X,Y) \leftarrow mother(X,Y)$$

Program reformulation is often necessary if the ILP learner enforces meta-constraints, such as a maximum number of clauses in a solution, or a maximum number of literals in the body of a clause.

The idea behind bias shift is to introduce predicates to make the target theory learnable. For example, consider the following example.

**Example 2.3.**

$$B = \begin{cases} parent(amy, amelia). \\ parent(gavin, amelia). \end{cases}$$

$$E^+ = \begin{cases} father(gavin, amelia). \end{cases}$$

$$E^- = \begin{cases} father(amy, amelia). \end{cases}$$

In this example, it is impossible to construct a hypothesis which covers the positive example but not the negative example using only the predicates in the background knowledge and examples. To learn a hypothesis consistent with the examples, an ILP learner must introduce a predicate symbol $m/1$ and the fact $m(gavin)$ to learn a hypothesis of the form $father(X,Y) \leftarrow m(X)$, where $m/1$ can be seen as inventing a *male/1* predicate.

### *2.3. Meta-interpretive learning*

We now briefly describe MIL [14,15], an ILP framework which supports predicate invention and upon which the implementation Metagol$_{pi}$ is based.

MIL is a form of ILP based on an adapted Prolog meta-interpreter. Given definite background knowledge $B$ and positive and negative unit clauses representing positive and negative examples $E^+$ and $E^-$, a MIL learner aims to return a definite hypothesis $H$ such that $B,H \models E^+$ and $B,H \not\models E^-$. The search for hypothesis is delegated to a meta-interpreter. However, whereas a standard Prolog meta-interpreter attempts to prove a goal by repeatedly fetching first-order clauses whose heads unify with given goals, a MIL meta-interpreter additionally attempts to prove a goal by repeatedly fetching higher-order metarules whose heads unify with given goals. The resulting meta-substitutions are saved in an abduction store which can be re-used in later proofs. Following the proof of a set of goals, a hypothesis is formed by projecting the meta-substitutions onto their corresponding metarules, allowing for a form of ILP which supports predicate invention and the learning of recursive theories. Completeness of SLD resolution ensures that all hypotheses consistent with the examples can be constructed. Moreover, unlike many ILP systems, only hypotheses consistent with all examples are considered [15]. The metarules are a form of declarative bias [3], and are defined separately from the meta-interpreter as part of the background knowledge. Figure 1 displays a small sample of metarules.

To demonstrate this technique, suppose that the background knowledge consists of the single ground atom *parent(alice,bob)* and the single *inverse* metarule (Figure 1), and that our goal is the ground atom *child(bob,alice)*. To prove this goal, a MIL learner fetches the *inverse* metarule and applies the meta-substitution $\theta = \{P/child\}$ to unify the head of the metarule with the goal. The unbound existential higher-order variable $Q$ in the metarule is then bound to *parent* predicate symbol in the predicate signature to form the meta-substitution $\theta\prime = \{P/child, Q/parent\}$. The ground atom *metasub(inverse,child,parent)*, representing the meta-substitution $\theta\prime$, is saved in an abduction store, and the learner continues the proof by attempting to prove the body of

| Name | Metarule |
|---|---|
| Base | $P(x,y) \leftarrow Q(x,y)$ |
| Inverse | $P(x,y) \leftarrow Q(y,x)$ |
| Precon | $P(x,y) \leftarrow Q(x),R(x,y)$ |
| Postcon | $P(x,y) \leftarrow Q(x,y),R(y)$ |
| Chain | $P(x,y) \leftarrow Q(x,z),R(z,y)$ |
| TailRec | $P(x,y) \leftarrow Q(x,z),P(z,y)$ |

**Figure 1.** Selection of metarules. The uppercase letters $P$, $Q$, and $R$ denote existentially quantified higher-order variables. The lowercase letters $x$, $y$, and $z$ denote universally quantified first-order variables.

the metarule. Once a proof is complete, the ground atom *metasub(inverse,child,parent)*, saved in the abduction store, is projected onto the corresponding metarule to obtain the inductive hypothesis $child(X,Y) \leftarrow parent(Y,X)$.

In MIL, predicate invention is implemented by adding Skolem constants representing new predicate symbols to the predicate signature. These can then be used as substitutes for the existentially quantified variables in metarules.

## 3. Experiments

This section describes experiments to investigate whether predicate invention can compensate for missing background predicates. We test the following null hypothesis.

**Null hypothesis** Predicate invention cannot compensate for missing background predicates.

### *3.1. Materials*

To test the null hypothesis, we compare the following three ILP systems.

- Metagol$_{pi}$[1] [15] is a MIL implementation based on an adapted Prolog meta-interpreter for a fragment of dyadic definite clause logic. This implementation supports predicate invention. Specifically, the implementation uses iterative deepening to ensure that the first hypothesis returned contains the minimal number of clauses, and at each depth $d$, Metagol$_{pi}$ introduces a new predicate symbol to the predicate signature.
- Metagol$_{nopi}$ is the same as Metagol$_{pi}$ except that the predicate invention feature is disabled.
- Progol5 [12] is a popular ILP system which does not perform predicate invention. Inverse Entailment is used with mode declarations to derive the most-specific clause within the mode language which entails a given example. This clause is used to guide a refinement-graph search by performing an admissible A*-like search, guided by compression, over clauses which subsume the most specific clause.

We conduct the experiments using the following two datasets.

[1] Metagol$_{pi}$ is named Metagol$_D$ in [15], but we have used an alternative name for clarity

1. Hinton's kinship data [7] contains 12 dyadic predicates and 104 examples.
2. Custom kinship dataset contains 21 dyadic predicates and 154 examples.

We also conducted experiments using a robot planning dataset. However, both Metagol$_{nopi}$ and Progol failed to learn solutions in almost every scenario with missing background predicates. By contrast, Metagol$_{pi}$ was able to learn solutions. This is discussed in Section 4.2.

### *3.2. Methods*

The experimental method is as follows. For each kinship relation in the dataset (the target predicate), we learn a hypothesis using the complete background knowledge. To do this, we randomly select $m$ training examples from the set $\{2,4,6,8,10\}$, half positive and half negative, of each target predicate and perform leave-one-out-cross-validation. Predictive accuracies are averaged over each target predicate over 50 trials. To explore the effect of missing background predicates, we repeat the aforementioned procedure, but with incomplete background knowledge. Specifically, for each $p$ in the sets $\{1,\ldots,12\}$ and $\{2,4,6,\ldots,20\}$ (datasets 1 and 2 respectively), we randomly select $p$ predicates to be removed from the background knowledge.

### *3.3. Results*

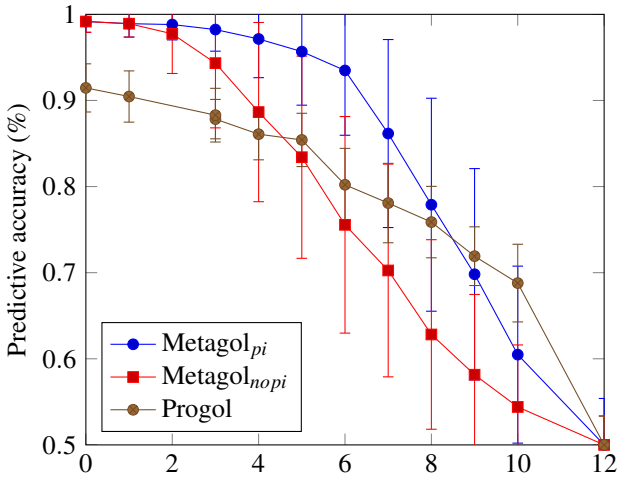


**Figure 2.** Predictive accuracies when learning kinship relations on Hinton's kinship dataset (dataset 1).

Figures 2 and 3 show that Metagol$_{pi}$ outperforms Metagol$_{nopi}$ on both datasets, except when all background predicates are present. In addition, the results show that Metagol$_{pi}$ generally outperforms Progol in terms of predicate accuracies, except in the case of few background predicates. Therefore, the null hypothesis is rejected. The results displayed are for training sizes of $m = 10$. Results for other values of $m$ are consistent but are omitted for brevity.

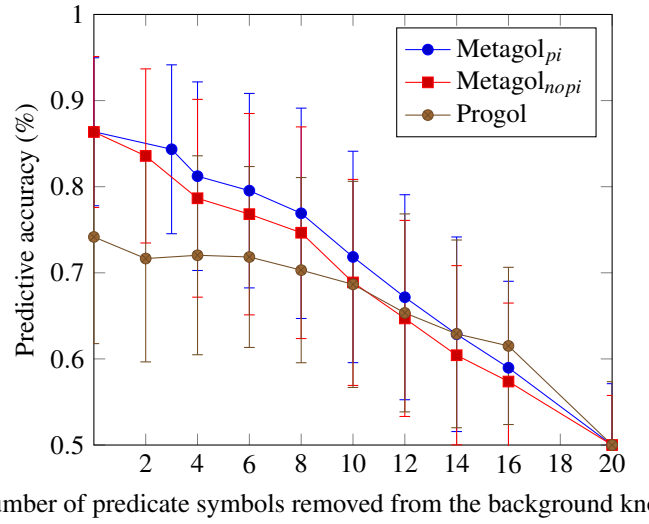


**Figure 3.** Predictive accuracies when learning kinship relations on the Custom kinship dataset (dataset 2).

It is noteworthy that in Figure 2 Metagol$_{pi}$ maintains *respectable* predictive accuracies with only half of the original background predicates, i.e. the predictive accuracies do not decrease in proportion to the decrease in background predicates. In Section 5 we discuss the implications of this.

## 4. Discussion

The rejection of the null hypotheses suggests that Metagol$_{pi}$ can compensate for missing background predicates through predicate invention. We discuss this using two scenarios: learning great-great-grandparent in dataset 2 of the experiments and learning robot plans.

### *4.1. Learning great-great-grandparent*

To illustrate how predicate invention can compensate for missing background predicates, consider learning the great-great-grandparent *(gggparent/2)* kinship relation using only the *mother/2* and *father/2* predicates. For this task Metagol$_{pi}$ learns the following definition.

```
gggparent(X,Y) :- p2(X,Z), p1(Z,Y).
p2(X,Y) :- p1(X,Z), p1(Z,Y).
p1(X,Y) :- father(X,Y).
p1(X,Y) :- mother(X,Y).
```

Metagol$_{pi}$ invents both the *parent* (p1/2) and *grandparent* (p2/2) predicates given only examples of the *gggparent/2* predicate. By contrast, learning a consistent *gggparent/2* hypothesis without predicate invention would require a solution with 16 clauses, which is often either intractable or prohibited by the meta-constraints.

*4.2. Learning robot plans*

As a second illustration, imagine a robot in a two-dimensional space. The robot can perform six dyadic actions: *left, right, forwards, backwards, grab* and *drop*. The task is to learn how to move a ball from a start position (1,1) to an end position (3,3). Figure 4 shows a solution to this problem learned by Metagol$_{pi}$ where *s1, s2*, and *s3* are invented high-level actions.

```
move(X,Y) :- s3(X,Z), drop(Z,Y).
s3(X,Y) :- grab(X,Z), s2(Z,Y).
s2(X,Y) :- s1(X,Z), s1(Z,Y).
s1(X,Y) :- forwards(X,Z), right(Z,Y).
```

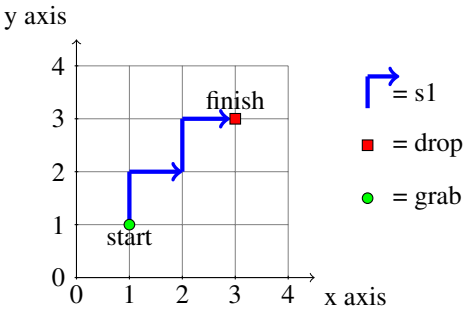


**Figure 4.** Robot plan learned by Metagol$_{pi}$ to move ball when supplied with complete background knowledge.

Now suppose that the actions *right*, *forwards*, and *drop* are removed from the background knowledge, i.e. the robot can no longer perform those actions. In this scenario, how can a robot complete the task? Figure 5 shows a solution for this problem learned by Metagol$_{pi}$ which uses the extra invented predicate *s4* which inverts the invented high-level action *s3*, thus allowing the actions *left*, *backwards*, and *grab* to indicate their inverse actions *right*, *forwards*, and *drop*. In this scenario, a single invented predicate has, effectively, replaced three background predicates. Neither Metagol$_{nopi}$ nor Progol learn a solution under these conditions.

```
move(X,Y) :- grab(X,Z), s4(Z,Y).
s4(X,Y) :- s3(Y,X).
s3(X,Y) :- s2(X,Z), s1(Z,Y).
s2(X,Y) :- grab(X,Z), s1(Z,Y).
s1(X,Y) :- left(X,Z), backwards(Z,Y).
```

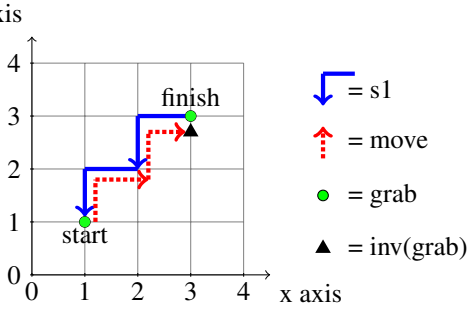


**Figure 5.** Robot plan learned by Metagol$_{pi}$ to move ball when supplied with incomplete background knowledge.

## 5. Conclusions and further work

In this paper, we have investigated whether predicate invention can compensate for missing background predicates. Our results show that as the number of background predicates

decrease, the MIL implementation Metagol$_{pi}$ outperforms Metagol$_{nopi}$ and Progol in terms of predictive accuracies, indicating that predicate invention can compensate for incomplete background knowledge.

We have, thus far, implicitly assumed that incomplete background knowledge is due to error or other unavoidable reason. Our results, however, suggest a possible motivation for purposely removing background predicates. In [8] it was shown that for a MIL learner, the number of programs of size $n$ which can be built from $p$ predicate symbols and $m$ metarules in the $H^2_2$ hypothesis space[2] is $O(p^{3n}m^n)$. Therefore, if working with many background predicates, then it might actually be preferable to purposely remove some predicates to reduce the hypothesis space to make the problem more tractable, whilst maintaining *respectable* predictive accuracies. This suggestion is in agreement with the Blumer bound [1], which says that a smaller hypothesis space leads to higher predictive accuracies and lower learning times compared to a larger hypothesis space, assuming that the target theory or its approximations are in both spaces.

### *5.1. Future work*

We have empirically shown that predicate invention can compensate for incomplete data and in future work we intend to show this theoretically.

In some ways, purposely removing background predicates is analogous to dimensionality reduction, widely applied in other forms of machine learning [20], but which has so far been neglected in ILP research [5]. In [2], the authors used Plotkin's clausal reduction algorithm [19] to logically minimise a maximal set of metarules in a fragment of dyadic definite clause logic. It seems reasonable to ask whether we can apply a similar technique to determine whether background predicates are redundant with respect to the metarules using Plotkin's algorithm. This involves applying the encapsulation theorem in [2] to convert first-order background primitives to higher-order clauses. Developing this theory is left for future work.

One limitation of this work is the relatively small datasets used in the experiments. Future work should include experiments in substantial domains, such as the mutagenesis dataset [21].

We also want to investigate whether the ability to compensate for missing background predicates through predicate invention is specific to the MIL framework, or whether other ILP learners which perform predicate invention achieve similar results, and empirical comparisons with other ILP systems that perform predicate invention, such as CIGOL [13], is left for future work.

## Acknowledgements

The first author acknowledges the support of the BBSRC and Syngenta in funding his PhD Case studentship. The second author would like to thank the Royal Academy of Engineering and Syngenta for funding his present five-year Research Chair.

[2] $H^j_i$ consists of definite function-free logic programs with predicates of adicity at most $i$ and with at most $j$ atoms in the body of each clause

## References


[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[2] Andrew Cropper and Stephen Muggleton. Logical minimisation of metarules within meta-interpretive learning. In *Proc. of the 24th International Conference on Inductive Logic Programming*, 2014. To appear.

[3] L. De Raedt. Declarative modeling for machine learning and data mining. In *Proceedings of the International Conference on Algorithmic Learning Theory*, page 12, 2012.

[4] Sašo Džeroski. Handling imperfect data in inductive logic programming. In *Proceedings of the Fourth Scandinavian Conference on Artificial Intelligence—93*, SCAI93, pages 111–125, Amsterdam, The Netherlands, The Netherlands, 1993. IOS Press.

[5] Johannes Fürnkranz. *Dimensionality reduction in ILP: A call to arms*. Citeseer, 1997.

[6] Zoubin Ghahramani and M.I. Jordan. Learning from incomplete data. Technical report, Lab Memo No. 1509, CBCL Paper No. 108, MIT AI Lab, 1995.

[7] G E Hinton. Learning distributed representations of concepts. *Artificial Intelligence*, 40:1–12, 1986.

[8] D. Lin, E. Dechter, K. Ellis, J.B. Tenenbaum, and S.H. Muggleton. Bias reformulation for one-shot function induction. In *Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, Amsterdam, 2014. IOS Press. In Press.

[9] Chunnian Liu and Ning Zhong. Rough problem settings for inductive logic programming. In *New Directions in Rough Sets, Data Mining, and Granular-Soft Computing*, pages 168–177. Springer, 1999.

[10] Benjamin M Marlin. *Missing data problems in machine learning*. PhD thesis, University of Toronto, 2008.

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

[12] S.H. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[13] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[14] S.H. Muggleton, D. Lin, N. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.

[15] S.H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 2015. Published online: DOI 10.1007/s10994-014-5471-y.

[16] Stephen Muggleton. Predicate invention and utilization. *Journal of Experimental & Theoretical Artificial Intelligence*, 6(1):121–130, 1994.

[17] Stephen H Muggleton, Jianzhong Chen, Hiroaki Watanabe, Stuart J Dunbar, Charles Baxter, Richard Currie, José Domingo Salazar, Jan Taubert, and Michael JE Sternberg. Variation of background knowledge in an industrial application of ilp. In *Inductive Logic Programming*, pages 158–170. Springer, 2011.

[18] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[19] G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, August 1971.

[20] David Skillicorn. *Understanding complex datasets: data mining with matrix decompositions*. CRC press, 2007.

[21] A. Srinivasan, S. H. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: Ilp experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming, volume 237 of GMD-Studien*, pages 217–232, 1994.

[22] Ashwin Srinivasan, S Muggleton, and RD King. Comparing the use of background knowledge by inductive logic programming systems. In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 199–230. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[23] Irene Stahl. Predicate invention in ilp an overview. In PavelB. Brazdil, editor, *Machine Learning: ECML-93*, volume 667 of *Lecture Notes in Computer Science*, pages 311–322. Springer Berlin Heidelberg, 1993.