Relational Knowledge Extraction from Attribute-Value Learners

Manoel V. M. França[1], Artur S. D. Garcez[2], and Gerson Zaverucha[3]

1,2 **Department of Computing**
School of Informatics, City University London
EC1V 0HB London, United Kingdom
manoel.franca.1,aag@city.ac.uk

3 **Programa de Engenharia de Sistemas e Computação**
COPPE, Universidade Federal do Rio de Janeiro
21941-972 Rio de Janeiro, Brazil
gerson@cos.ufrj.br

Abstract

Bottom Clause Propositionalization (BCP) is a recent propositionalization method which allows fast relational learning. Propositional learners can use BCP to obtain accuracy results comparable with Inductive Logic Programming (ILP) learners. However, differently from ILP learners, what has been learned cannot normally be represented in first-order logic. In this paper, we propose an approach and introduce a novel algorithm for extraction of first-order rules from propositional rule learners, when dealing with data propositionalized with BCP. A theorem then shows that the extracted first-order rules are consistent with their propositional version. The algorithm was evaluated using the rule learner RIPPER, although it can be applied on any propositional rule learner. Initial results show that the accuracies of both RIPPER and the extracted first-order rules can be comparable to those obtained by Aleph (a traditional ILP system), but our approach is considerably faster (obtaining speed-ups of over an order of magnitude), generating a compact rule set with at least the same representation power as standard ILP learners.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving

Keywords and phrases Relational Learning, Propositionalization, Knowledge Extraction

Digital Object Identifier 10.4230/OASIcs.ICCSW.2013.35

1 Introduction

Relational learning can be described as the task of learning a first-order logic theory from examples [10, 3]. Inductive Logic Programming (ILP) [15, 17] performs relational learning either directly by manipulating first-order rules or through a method called propositionalization [13, 22], which brings the relational task down to the propositional level by representing subsets of relations as features that can then be used as attributes. In comparison with full ILP, propositionalization normally exchanges accuracy for efficiency [11], as it enables the use of fast attribute-value learners such as rule learners [2], but could lose information in the translation of first-order rules into features. Bottom Clause Propositionalization (BCP) [5] is a recent propositionalization method which allows fast relational learning and also allows propositional learners to obtain accuracy results on par with Inductive Logic Programming (ILP) learners, although differently from ILP learners, what has been learned is not possible to be represented in first-order.

In this paper, we introduce a novel algorithm for consistent extraction of first-order rules from propositional rule learners, when dealing with data propositionalized with BCP. Bottom clauses are variablized first-order clauses that are used as boundaries in ILP hypothesis search space, firstly introduced by Progol [14]. Given an ILP dataset, bottom clauses are built from one positive example e, background knowledge BK (a set of clauses that describe what is known) and language bias L (a set of clauses that define how clauses can be built). A bottom clause is the most specific clause (with most literals) that can be considered a candidate hypothesis. BCP uses bottom clauses for propositionalization because they carry semantic meaning, and because bottom clause literals can be used directly as features in a truth-table, simplifying the feature extraction process [16, 4]. BCP extends Progol's bottom clause generation algorithm to deal with negative examples and it keeps track of a hash table which is responsible to map each constant found during bottom clause generation to an unique variable, for each example. This hash table is one of the key differences between BCP and other propositionalization methods such as RSD [22], SINUS [10] and RELAGGS [12], and is used by our approach to transform propositional rules learned from data which was propositionalized with BCP (which we will refer in this work simply as BCP-rules) to consistent and accurate relational (first-order) rules. A theorem then shows that the extracted first-order rules are consistent with their propositional version.

Our methodology is evaluated using BCP for propositionalization and the propositional rule learner RIPPER on four Alzheimer datasets [9] and we have used three metrics to present our results: standard classification accuracy (the percentage of correctly classified examples, among all tested examples), runtime (total time taken to complete the experiment, from training to testing), and rule size (we define rule size as the number of body and head literals in the entire induced theory). Results show that although information loss is expected when dealing with propositionalization methods [22, 10, 5], the accuracy of both RIPPER and the first-order rules can be comparable with Aleph [20], a traditional ILP system, in some cases, while being considerably faster and generating rules with lower size. The rules generated by our algorithm also have more representational power, by being able to represent disjunctions and negation as failure.

The remainder of this paper is as follows: in Section 2, we review BCP and the propositional rule learner RIPPER. In Section 3, we introduce our contribution: an algorithm for extracting relational rules from BCP-rules. Our empirical results with regard to classification accuracy, runtime and rule size in comparison with RIPPER and Aleph is shown in Section 4, and in Section 5, we conclude and discuss directions for future work.

2 Background

In this section, the key methods and algorithms used in our work are introduced: Bottom Clause Propositionalization and RIPPER.

Bottom Clause Propositionalization [5] is a logic-based propositionalization method which takes advantage of Progol's bottom clause generation algorithm and the previous work from [4], which shows that bottom clause literals can be used as propositional features. In [5], BCP managed to achieve comparable results with Aleph in a number of ILP datasets, even though propositionalization methods incur information loss. Additionally, BCP was faster and also obtained better accuracy and runtime results when compared to RSD. One problem with the obtained results, though, was that all tests were done in the propositional level. We investigate in this paper how to bring those results back to first-order.

BCP has two steps: bottom clause generation and attribute-value mapping. In the first step, each example is given to Progol's bottom clause generation algorithm [21] to create a corresponding bottom clause representation. To do so, a slight modification is needed to allow the same *hash* function to be shared among all examples, in order to keep consistency between variable associations (i.e., to ensure that variable associations are done in the same way, for different bottom clauses), and to allow negative examples to have bottom clauses as well; the original algorithm deals with positive examples only. The generation algorithm has a single parameter, *depth*, which is the *variable depth* of the bottom clause generation process. A more detailed description of the modified version of Progol's bottom clause generation algorithm can be found in [5].

To illustrate BCP's bottom clause generation, consider the well-known family relationship [15] ILP example: $BK = \{$*mother(mom1, daughter1)*, *wife(daughter1, husband1)*, *wife(daughter2, husband2)*$\}$, positive example *motherInLaw(mom1, husband1)*, and negative example *motherInLaw(daughter1, husband2)*. If the modified bottom clause generation algorithm is executed with $depth = 1$ on the positive and negative examples shown above, it generates the following training set: $S_\perp = \{motherInLaw(A,B)$:- $mother(A,C), wife(C,B);$ $\sim motherInLaw(A,B)$:- $wife(A,C)\}$.

After the creation of the $S_\perp$ set, each bottom clause inside $S_\perp$ is converted into a binary vector v_i, $0 \leq i \leq n$, according to the presence or absence of each found literal inside $S_\perp$, where n is the number of distinct literals inside $S_\perp$.

RIPPER [2] is a well-known propositional rule learner which can be considered the propositional version of the FOIL first-order theory induction algorithm [8] in the sense that it also performs a covering-based algorithm to choose literals to build its theory, using information gain as search heuristic. RIPPER's focus is to tackle noisy data and achieve competitive results with regard to Quinlan's propositional tree-learner C4.5 [18]. RIPPER extends its predecessor IREP [6], by improving its information gain heuristic and its stopping criteria (this improvement was named IREP*), and uses IREP* multiple times, to perform different parts of the learning task. Those parts are: IREP* is used once to obtain an initial rule set, covering part of the positive examples; The rules are optimized with regard to redundancy/consistency; and IREP* is used again to cover the remaining positive examples.

RIPPER has been shown in [2] to generate rules with better performance than C4.5's decision trees and to be efficient (fast and accurate) on large and noisy datasets. RIPPER's ability to generate good rules when dealing with large and noisy datasets is the reason it is chosen to process data which was propositionalized with BCP: bottom clauses can be considerably large, possibly having infinite size [14], and a learner which can deal with large number of noisy features is better suited to deal with BCP.

3 Extracting Relational Knowledge from BCP-Rules

In this section, our algorithm for generating first-order rules from BCP-rules is introduced. It is important to notice that the described methodology below does not require a specific attribute-value learner: any propositional learner which is able to generate logic rules to describe what has been learned, e.g. decision tree learners, rule learners and graph learners, are all able to be used for the proposed rule extraction.

As a first step of our approach, we explain hereafter how BCP-rules are generated. Firstly, BCP is applied in the examples set, generating a bottom clause set $S_\perp$, as shown in Section 2. Then, attribute-value learning takes place. In this work, we have chosen RIPPER due to the advantages described in Section 2, but any learner that can generate rules can be used, although further investigation is required.

By using RIPPER on data propositionalized with BCP, a set of rules is created. From an ILP point of view, those rules do not necessarily obey variable chaining properties or any kind of language bias restrictions: each feature (i.e. each distinct bottom clause atom) is seen as propositional features by RIPPER and thus, further processing needs to be done in order to treat it as first-order. As an example, consider the following propositionalized dataset:

$$\begin{aligned} S_\perp = \{motherInLaw(A,B) &\ :-\ mother(C,B), wife(C,D); \\ motherInLaw(A,B) &\ :-\ mother(A,C), wife(C,B); \\ \sim motherInLaw(A,B) &\ :-\ wife(C,B), parents(C,B,D), dad(E,F)\}. \end{aligned} \tag{1}$$

From this dataset, one possible rule generated by RIPPER (containing features from positive examples and negated features from negative examples), in Prolog format, could be:

$$R_\perp = \{motherInLaw(A,B) :- mother(A,C), wife(C,D), not(dad(E,F))\}. \tag{2}$$

The first point worth noticing regarding $R_\perp$ is that it can represent the *absence* of a BCP feature, e.g. $not(dad(E,F))$, which is equivalent to negation as failure [7] and shows that the rules we generate have more representational power. The second point is that there is a problem with the generated BCP-rule $R_\perp$, if it is treated directly as first-order: the variables of $dad(E,F)$ are not present in any other body or head atom (from now on, we will refer to those variables as *unconstrained variables*). This is possible to happen due to the fact that all atoms are seen as features generated by BCP, thus not taking into consideration the language bias. If $R_\perp$ is used as theory to infer unseen first-order data, as long as there is at least one $dad/2$ ground atom[1] inside the background knowledge, $dad(E,F)$ would always be true and thus, $not(dad(E,F))$ would always be false and $R_\perp$ would always be false as well, thus limiting the generalization capabilities of $R_\perp$. In order to solve this issue, after generating BCP-rules using RIPPER and obtaining a set of BCP-rules $R_\perp$, all unconstrained variables need to be removed from $R_\perp$ before treating it as a first-order theory.

The process of extracting first-order rules from BCP-rules can be divided into three steps: *unconstrained variables search*, *unconstrained variables replacing* and *first-order filtering*. In the first step, *unconstrained variables search*, a search is done in the BCP-rules to find literals with unconstrained variables (i.e., finding all occurrences of literals such as $not(dad(E,F))$ in $R_\perp$ above). We detect unconstrained variables from the rightmost literal to the leftmost one, by verifying if the variable being checked, belonging to a body literal l_i, appears on any other body literal in $\{l_j | j < i\}$. For each BCP-rule $r \in R_\perp$, we store unconstrained variables (and the literals where they were found) to be used in the next step of our algorithm, *unconstrained variables grounding*.

As an example, let us use the $R_\perp$ defined in (2) as input for *unconstrained variables search*. Firstly, unconstrained variables are searched in the single clause of $R_\perp$, from the right to the left. In the rightmost literal, $not(dad(E,F))$, two unconstrained variables are found: E and F. Because of that, both variables and the literal where they were found are stored. After advancing to the next rightmost literal, one more unconstrained variable is found: D. Thus, D is stored for the next part of our algorithm, together with the literal $wife(C,D)$ where it was found. Note that the other variable, C, is not included, since it appears in $mother(A,C)$ and thus, it is not unconstrained. Since no more unconstrained variables can be found, the first step of our extraction algorithm comes to an end with the following variables/literals stored: $M = \{E, F, not(dad(E,F)); D, wife(C,D)\}$.

[1] Ground atoms are atoms which does not contain variables, only constants.

After that, *unconstrained variables replacing* comes into place. All the stored variables are replaced using the *hash* table generated during BCP, as mentioned on Section 2, in order to eliminate all unconstrained variables which were found in the previous steps. To illustrate that, let us continue our family relationship example. Three variables have been flagged as unconstrained: E and F, from the literal $not(dad(E, F))$, and D, from the literal $wife(C, D)$. Assume the same propositionalized examples set $S_\perp$ shown in (1), that generated the BCP-rules set $R_\perp$. Since the body of the first example of $S_\perp$ contains one variable of M (which is D), the second example does not contain any of the variables mapped in M, and the third example contains all three variables inside M, which are D, E and F (totalizing two occurrences of D and one occurrence of both E and F on $S_\perp$), BCP's *hash* must have two entries for D, one entry for E and one entry for F. Let us assume that those entries are $\{D/husband1\}$, $\{D/daughter1\}$, $\{E/mom1\}$ and $\{F/daughter1\}$.

As explained earlier, as long as a BCP-propositionalized example contains a literal lexically identical to a literal from another example, both will be considered to have the feature represented by that literal. In the case of the feature $wife(C, D)$, for instance, even though different examples have different *hash* mappings for C and D, the presence or not of $wife(C, D)$ is what is considered for propositional learning. This suggests that the rule that defines the truth-value of a feature is a *disjunction over all observed mappings (unifications) in the training set during BCP propositionalization*. Theorem 2 below shows that for each BCP feature, a disjunction over all possible unifications of a feature with grounding operators over all examples is semantically equivalent to the feature itself.

▶ **Definition 1.** Let $e \in E$ be an example of an dataset E, propositionalized with BCP. Also, let f be a BCP feature, let U be the set of all unconstrained variables which can be found inside f, having size k, and let $hash_e$ be the variable/constant mapping generated for example e during BCP. The *grounding unifier* θ_e^f for a feature f with regard to an example e is defined as $\theta_e^f = \{v^1/c^1, v^2/c^2, \cdots, v^k/c^k\}$, where $v_i \in U, 1 \leq i \leq k$ is an unconstrained variable and c_i is the constant which is mapped to v_i, according to $hash_e$. If $k = 0$, $\theta_e^f = \emptyset$.

▶ **Theorem 2.** *Let E be an example set, propositionalized with BCP and having size n, and f be one of the generated features with BCP when applied to E. Also, let $v(f)$ be a valuation function, which associates a boolean truth-value for a feature f. Then, $v(f) \equiv v(f\theta_{e_1}^f) \vee v(f\theta_{e_2}^f) \vee \cdots \vee v(f\theta_{e_n}^f)$, where $\{e_1, e_1 \cdots e_1\} \subset E$ and $f\theta_{e_i}^f$ is the unification of feature f with a grounding unifier $\theta_{e_i}^f$, $1 \leq i \leq n$.*

Proof. Proof by contradiction. Suppose that there exists a feature f and examples $e_i \in E$, $1 \leq i \leq n$, where $v(f) \not\equiv v(f\theta_{e_1}^f) \vee v(f\theta_{e_2}^f) \vee \cdots \vee v(f\theta_{e_n}^f)$ holds. There are two case scenarios that makes this equation true:

- There exists a feature f with truth-value *true*, but all possible unifications of f with ground unifiers $\theta_{e_i}^f, 1 \leq i \leq n$, are false. If f appears in a rule, it must have been found in at least one bottom clause generated with BCP from an example $e \in E$. Then, there exists one set of unifications $\{v/c\}$ in $hash_e$, one for each v inside f, which makes $f\{v/c\}$ true. If this unifier is used as θ_e^f, then at least one member of the disjunction is true.
- There exists a feature f which is false, but at least one possible unification $\theta_{e_i}^f$ of f, $1 \leq i \leq n$, with ground unifier $\theta_{e_i}^f$, is true. Definition 1 ensures that f can be found inside e_i, otherwise $\theta_{e_i}^f$ would not exist. Thus, if $\theta_{e_i}^f$ is a valid unifier for e_i, it also needs to be a valid unifier for f.

◀

We now can solve the problem of BCP-rules having unconstrained variables by replacing them with disjunctions of grounding unifications (we call those unified BCP-rules *constrained BCP-rules*). We illustrate the second step of our algorithm by continuing our family relationship example. From the first step of our relational knowledge extraction algorithm, we have obtained a list of variables that are unconstrained and need to be replaced: E and F, from feature $not(dad(E, F))$, and D, from feature $wife(C, D)$. From (1), one example contains the feature $not(dad(E, F))$ and two contain feature $wife(C, D)$. Thus, $not(dad(E, F))$ is replaced by one grounded literal and $wife(C, D)$ is replaced by a disjunction of two grounded literals, by applying Theorem 2 and using the previously specified *hash* entries for those examples: $\{D/husband1\}$, $\{D/daughter1\}$, $\{E/husband2\}$ and $\{F/daughter1\}$. Those replacements are $\{not(dad(E, F)) \mapsto not(dad(husband1, daughter1))\}$ and $\{wife(C, D) \mapsto wife(C, daughter1) \vee wife(C, husband2)\}$ and thus, the resulting constrained BCP-rule set $R^C_\perp$ after replacing $R_\perp$ from (2) with the created grounded atoms (in prolog format) is

$$R^C_\perp = \{motherInLaw(A, B) : -mother(A, C), (wife(C, daughter1); wife(C, husband2)), \\ not(dad(husband1, daughter1))\}.$$

Lastly, in *first-order filtering*, we apply a modified version of the theory filtering algorithm T-reduce [20], a companion program to Aleph, in theory $R^C_\perp$. The original T-reduce algorithm is capable of removing rules that do not cover any first-order training example and rules that contribute negatively to the theory accuracy. We modified T-reduce to also to cut out redundant literals and literals that do not have variables on it. Literals without variables need to be removed for the same reason the unconstrained variables of $not(dad(E, F))$ need to be replaced: depending on the background knowledge, those literals are always true or always false, thus contributing negatively to the rule's ability to generalize. As an example, if our version of T-reduce is applied on $R^C_\perp$, assuming that $R^C_\perp$ is non-redundant (otherwise it would be removed by T-reduce), we obtain the final first-order theory $R^{FOL}_\perp = \{motherInLaw(A, B)$:- $mother(A, C), (wife(C, daughter1); wife(C, husband2))\}$, since $not(dad(husband1, daughter1))$ does not have variables on it.

To illustrate the whole process of extracting first-order rules from BCP-rules, our complete procedure is summarized in Algorithm 1. It receives as input a set of BCP-rules $R_\perp$ and outputs a set $R^{FOL}_\perp$ of extracted first-order rules.

Algorithm 1 First-order Rules Extraction from BCP-rules

```
1: $R^{FOL}_\perp = \emptyset$
2: Let $U$ be the set of unconstrained variables and their respective literals inside $R_\perp$
3: for each rule $r$ of $R_\perp$ do
4:     Apply Theorem 2 by using $U$ on $r$ to obtain a constrained clause $c_r$
5:     Apply (modified) T-reduce on $c_r$ to obtain a filtered clause $r_{treduce}$
6:     Check if $r_{treduce}$ contributes positively towards accuracy; if not, discard it
7:     Add $r_{treduce}$ to $R^{FOL}_\perp$, if it has not been discarded
8: end for
9: return $R^{FOL}_\perp$
```

4 Initial Results

In this section, we present the experimental methodology and initial results for our relational knowledge extraction algorithm. We show comparative results between the ILP system

Aleph, RIPPER when trained with BCP-data (we will refer to it as BCP+RIP$_{prop}$) and the extracted first-order rules from BCP+RIP$_{prop}$ (we will refer to it as BCP+RIP$_{FOL}$), using the methodology presented on Section 3. We have used the *Alzheimers* benchmark [9], which consists of four datasets: *Amine*, *Acetyl*, *Memory* and *Toxic*. The used experimental configurations on Aleph, RIPPER and BCP can be found on `http://soi.city.ac.uk/~abdz937/iccsw13Parameters.txt`.

We evaluate the results on three aspects: *standard accuracy*, *runtime* and *theory size*. We define standard accuracy as the percentage of correctly classified examples over test data; we define runtime as the total pre-processing, training and testing times for each system; and we define theory size as the total number of literals (body literals and head literals) in the learned theory. Our obtained results, averaged by 10-fold cross-validation for accuracy and theory size, and accumulated over all 10 fold with regard to runtime, are presented on Table 1. In the accuracy results, values in bold are the highest ones obtained between BCP+RIP$_{prop}$ and BCP+RIP$_{FOL}$ and the difference between them and the ones marked with asterisk (*) are statistically significant by two-tailed, paired t-test.

Table 1 Accuracy results (with standard deviation), runtimes and theory size measurements for the Alzheimers benchmark (accuracies in the first line; runtimes, theory size in the second line). The results accuracy difference between BCP+RIP$_{prop}$ and Aleph, on all four datasets, are not statistically significant. Between BCP+RIP$_{prop}$ and BCP+RIP$_{FOL}$, BCP+RIP$_{FOL}$ managed to obtain statistically comparable accuracy results with BCP+RIP$_{prop}$ in the first two datasets (*Alz-ami* and *Alz-ace*). Comparing directly BCP+RIP$_{FOL}$ with Aleph, BCP+RIP$_{FOL}$ also managed to obtain statistically comparable results on two datasets, *Alz-ami* and *Alz-ace*. Additionally, it can be seen that the our methodology is considerably faster than Aleph, obtaining an average speed-up over all four datasets of more than one order of magnitude, while generating smaller rules as well.

	Alz-ami	***Alz-ace***	***Alz-mem***	***Alz-tox***
Aleph	78.71(±5.25)	69.46(±4.6)	68.57(±5.7)	80.5(±4.83)
(baseline)	1:31:05, 36.1	8:06:06, 47.3	3:47:55, 45.7	6:02:05, 37.9
BCP+RIP$_{prop}$	73.35(±4.32)	**67.8**(±3.77)	**65.27**(±7.11)	**78.44**(±5.44)
	0:19:49/30	0:23:21/20.1	0:25:11/14.4	0:17:41/35.2
BCP+RIP$_{FOL}$	**77.73**(±4.57)	63.56(±5.06)	57.64 $*$ (±5.7)	66.45 $*$ (±6.93)
	0:21:59/30.4	0:26:39/18.7	0:28:45/13.8	0:20:57/18

5 Conclusion and Future Work

This paper has tackled the problem of accurately and consistently represent what has been learned with data previously propositionalized with BCP, but back to a relational format, by introducing a novel algorithm for consistent extraction of first-order rules from propositional rules described with BCP features. A theorem shows that the extracted first-order rules are consistent with their propositional version. Our results show that the presented methodology is promising and it can not only extract accurate rules from propositional learners which used BCP as propositionalization method, but it can also improve its accuracy in the process. Additionally, our approach is capable of generating first-order rules with disjunctions and negation as failure, which standard ILP inducers cannot, and our results show that our approach is considerably faster than Aleph (a speed-up of over an order of magnitude on average) and also generates smaller rules.

As future work, experiments on datasets with numerical (continuous) data, such as Carcinogenesis [19] and with very large data, such as CORA [1] are underway. Also, a new version of RIPPER which satisfies mode declarations, for learning with BCP, is being studied.

References

1 M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proc. ACM SIGKDD*, pages 39–48, New York, NY, USA, 2003.

2 W. W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart J. Russell, ed., *ICML*, pg. 115–123. Morgan Kaufmann, 1995.

3 L. De Raedt. *Logical and Relational Learning.* Cognitive Technologies. Springer, 2008.

4 F. DiMaio and J. W. Shavlik. Learning an Approximation to Inductive Logic Programming Clause Evaluation. In *ILP*, vol. 3194 of *LNAI*, pg. 80–97, 2004.

5 M. V. M. França, G. Zaverucha, and A. S. D. Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Mach. Learn.*, pg. 1–24, 2013.

6 J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In William W. Cohen and Haym Hirsh, ed., *ICML*, pg. 70–77. Morgan Kaufmann, 1994.

7 M. L. Ginsberg, editor. *Readings in nonmonotonic reasoning.* Morgan Kaufmann, San Francisco, CA, USA, 1987.

8 J. R. Quinlan and R. M. Cameron-Jones. FOIL: A Midterm Report. In *Proc. ECML*, pages 3–20. Springer, 1993.

9 R.D. King and A. Srinivasan. Relating chemical activity to structure: An examination of ILP successes. *New Generation Computing*, 13(3-4):411–434, 1995.

10 S. Kramer, N. Lavrač, and P. Flach. Relational Data Mining. chapter Propositionalization approaches to relational data mining, pg. 262–286. Springer, New York, NY, USA, 2000.

11 M. A. Krogel, S. Rawles, F. Železný, P. Flach, N. Lavrač, and S. Wrobel. Comparative Evaluation Of Approaches To Propositionalization. In *ILP*, vol. 2835 of *LNAI*, pg. 194–217. Springer, 2003.

12 M. A. Krogel and S. Wrobel. Facets of Aggregation Approaches to Propositionalization. pg. 30–39. Department of Informatics, University of Szeged, September 2003.

13 N. Lavrač and S. Džeroski. *Inductive logic programming: techniques and applications.* Ellis Horwood, 1994.

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

15 S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

16 S. Muggleton and A. Tamaddoni-Nezhad. QG/GA: a stochastic search for Progol. *Mach. Learn.*, 70:121–133, 2008.

17 S.H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, vol. 1228 of *LNAI.* Springer, 1997.

18 J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Francisco, CA, USA, 1993.

19 A. Srinivasan, R. D. King, S. H. Muggleton, and M. J. E. Sternberg. Carcinogenesis Predictions using ILP. *Proc. International Workshop on Inductive Logic Programming*, 1297(1297):273–287, 1997.

20 A. Srinivasan. The Aleph System, version 5. http://www.cs.ox.ac.uk/activities/machlearn/Aleph/aleph.html, June 2007. Last accessed on July/2013.

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

22 F. Železný and N. Lavrač. Propositionalization-based Relational Subgroup Discovery With RSD. *Machine Learning*, 62:33–63, 2006.