A Framework for Set-Oriented Computation in Inductive Logic Programming and its Application in Generalizing Inverse Entailment*

Héctor Corrada Bravo[1], David Page[2,1], Raghu Ramakrishnan[1], Jude Shavlik[1,2], and Vitor Santos Costa[2,3]

[1] Department of Computer Sciences
[2] Department of Biostatistics and Medical Informatics
University of Wisconsin-Madison, USA
[3] COPPE/Sistemas UFRJ, Brasil
{hcorrada,raghu,shavlik}@cs.wisc.edu
{page,vitor}@biostat.wisc.edu

Abstract. We propose a new approach to Inductive Logic Programming that systematically exploits caching and offers a number of advantages over current systems. It avoids redundant computation, is more amenable to the use of set-oriented generation and evaluation of hypotheses, and allows relational DBMS technology to be more easily applied to ILP systems. Further, our approach opens up new avenues such as probabilistically scoring rules during search and the generation of probabilistic rules. As a first example of the benefits of our ILP framework, we propose a scheme for defining the hypothesis search space through Inverse Entailment using multiple example seeds.

1 Introduction

The goal of Inductive Logic Programming (ILP) [1] is to autonomously learn first-order logic programs that model relational data. However, the current approach to ILP has limitations in its scalability and computational efficiency. Recent efforts extend ideas from relational database query optimization to this setting [2–6]. Along the same line, we present a new formulation of ILP that systematically exploits caching to achieve greater efficiency and flexibility, and present theoretical results that characterize it.

The fundamental building blocks for our approach are a new data structure and an extension operation for hypotheses that expose and exploit opportunities for caching the results of previous computation. This provides an immediate benefit by avoiding the redundant computation pervasive in the standard ILP search and score paradigm. Further, the extension operation is formulated as a set-oriented computational strategy defined in terms of (extended) relational

* Work was supported by Air Force Grant F30602-01-2-0571, DARPA ISTO Grant HR0011-04-0007 and a Ford Fellowship from the National Academy of Sciences.

database operations, facilitating the use of relational query-processing techniques in ILP systems.

Extensional representations of hypotheses are treated as first-class objects. Consequently, statistics derived from these objects are easily maintained, and can be used to define alternatives to guide the search process in ILP. For example, probabilistic methods for search [7, 3, 4, 8, 9] can directly use statistics derived from our new data structure for representing hypotheses.

Additionally, statistics derived from an extensional representation of hypotheses offer new avenues for learning a class of rules richer class than Horn clauses. For example, rules containing statements about aggregates [10], and rules containing probabilistic statements, such as statements about missing values [11], can be generated. While these extensions are beyond the scope of this paper, we investigate a scheme for restricting the hypothesis search space using Inverse Entailment based on a set of multiple seed examples. Our algorithm for Generalized Inverse Entailment offers flexibility and robustness in hypothesis space restriction, including alternative seed-coverage measures (which we study in this paper) and cost-based measures that can be readily obtained from our hypothesis representation.

Our main contributions are as follows:

(1) New data representation and extension-join operation (Section 3), with a discussion of potential benefits (Section 3.2).
(2) New set-oriented hypothesis generation framework, with proof of soundness and completeness with respect to inverse entailment under subsumption for the single-seed case (Section 4).
(3) Generalization of inverse entailment to the multiple-seeds case; extension of our hypothesis generation framework to this case; and a proof of soundness and completeness with respect to a generalized coverage measure (Section 5).

2 Mode-Restricted Languages

The ILP task consists of learning a logic program that models a dataset of ground facts, given as two disjoint sets of positive examples and negative examples. We are also given "background knowledge" in the form of additional facts or predicates defined as Horn clauses. The learned program is a set of Horn clauses that, when added to the background knowledge, entails as many of the positive example facts as possible while entailing as few of the negative example facts as possible. Each clause, or hypothesis, in the learned program is built from the predicates given in the background knowledge, and we assume they are functor-free Horn clauses. In this section we define the space of hypotheses we seek to represent.

We borrow the concept of user-specified "modes" that constrain the space of allowable hypotheses from the Aleph [12] and Progol [13] ILP systems.

Definition 1. *A mode is defined by* $(\mathrm{p/n}, \mathrm{B}, \mathrm{F})$ *where:* p *is an* n*-ary background predicate,* B *is the list of arguments of* p *specified to be* bound, F *is the list of arguments of* p *specified to be* free.

Further restrictions are that B and F are disjoint sets, every argument in the predicate is specified as either *bound* or *free*, and at least one *bound* argument is specified. (We do not include the case where modes specify arguments to be constants, but our results apply to this case as well.)

Definition 2. *A moded literal* $\mathrm{p}^*(\mathrm{A},\ldots,\mathrm{N})$ *is the adornment of a literal as specified by the binding pattern in a mode.*

As an example, let q be a background predicate. If $q(A,B)$ is a literal, and mode $(q/2,[1],[2])$ is defined, then $q^{bf}(A,B)$ is an allowed moded literal. The first argument of q is specified as bound, that is, an input argument in the usual ILP nomenclature, while the second is specified as free, that is, an output argument. In the rest of this paper, we treat arguments in literals as implicit when their details are not required, and use p^* to denote moded literals.

For convenience, we define the following operations on moded literals: let p^* be the moded literal specified by mode $(p/n, B, F)$, then: $\mathrm{bound}(p^*) = B$, $\mathrm{free}(p^*) = F$, $\mathrm{vars}(p^*) = B \cup F$, and $\mathrm{pred}(p^*) = p$.

Given a set of modes, we denote the set of allowable moded literals as $\mathcal{M}$. From now on, we assume every literal in a hypothesis is a moded literal and leave the adornment implicit when not needed. With this set of allowable moded literals the mode-restricted set of hypotheses can be defined recursively as follows:

Definition 3. *Given a set of allowable moded literals* $\mathcal{M}$*, and a target predicate* h*, the set* $\mathrm{H}(\mathcal{M},\mathrm{h})$ *of hypotheses allowable in the mode-restricted language is recursively defined as:* $H(\mathcal{M},h) = \{(h \leftarrow \mathrm{true.})\} \cup \{(h \leftarrow r_1,\ldots,r_n,p.) : (h \leftarrow r_1,\ldots,r_n.) \in H(\mathcal{M},h), p \in \mathcal{M},$ *and* $\mathrm{bound}(p) \subseteq \mathrm{Vars}(h \leftarrow r_1,\ldots,r_n.)\}$

The set of variables $\mathrm{Vars}(h \leftarrow r_1,\ldots,r_n.)$ of a hypothesis is the union of the variable sets of its literals. That is, $\mathrm{Vars}(h \leftarrow r_1,\ldots r_n.) = \mathrm{vars}(h) \bigcup_{i=1}^{n} \mathrm{vars}(r_i)$. The given positive facts we want to model are instances of the target predicate h. For example, let r and q be background predicates and let there be modes that specify the adornments r^{bf} and q^{fb}. Then, $\hat{h} = (h(A) \leftarrow r^{bf}(A,B), q^{fb}(C,B).) \in H(\mathcal{M},h)$, while $\hat{h} = (h(A) \leftarrow r^{bf}(A,B), q^{fb}(B,C).) \notin H(\mathcal{M},h)$.

The set $H(\mathcal{M},h)$ in Definition 3 is the set of hypothesis we want to capture using our representation.

3 The WILD Representation

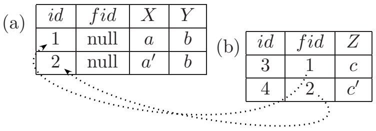
We seek to represent hypotheses in $H(\mathcal{M},h)$ in such a way that, intuitively, the result of operating on a hypothesis is reused when operating on an extension of the hypothesis. For instance, when measuring the coverage of a hypothesis, the substitution found in proving that a hypothesis covers an example contains bindings which could potentially make an extension of the hypothesis cover the same example. Our representation should reuse those bindings when measuring the coverage of the extended hypothesis.

Another objective is that useful statistics regarding a hypothesis should be easy to derive and maintain from the representation of hypotheses. Continuing with the example, we want the coverage of a hypothesis for a given example to be easily recoverable from our data structure. However, we are interested in maintaining statistics that are useful for measures other than coverage. We now define a data representation and extension operation for hypotheses that meet these goals.

Definition 4. *Hypothesis* $\hat{\mathrm{h}} \in \mathrm{H}(\mathcal{M}, \mathrm{h})$ *is represented by the pair* $\langle \hat{\mathrm{h}}, \mathrm{t} \rangle$*, where* t *is a database table. t has schema* $t[id, fid, A, \ldots, N]$*, where* id *is a unique (across all existing tables) row identifier;* fid *is the unique identifier of the 'parent' row; and* $\mathrm{A}, \ldots, \mathrm{N}$ *are variable names appearing in* $\hat{h}$*.*

Each row of t is a binding that makes $\hat{\mathrm{h}}$ cover a fact e in a set of seeds E. The schema of t serves to share common subsets of bindings between hypotheses by its use of the *id* and *fid* fields. Given a set of seeds, an initial table is built where *fid* is *null* and each seed is represented by one row.

As an example, consider hypothesis $\hat{h}_1 = (h(X, Y) \leftarrow q^{bf}(X, Z).)$, along with seed table t_0 shown in Fig. 1(a), built for the seed set $\{h(a, b), h(a', b)\}$. Let the facts $q(a, c)$ and $q(a', c')$ be in the table for background predicate q. $\hat{h}_1$ is represented as the intensional/extensional pair $\langle h(X, Y) \leftarrow q^{bf}(X, Z)., t_1 \rangle$, where t_1 is shown in Fig. 1(b).



(a)

id	*fid*	*X*	*Y*
1	null	a	b
2	null	a'	b

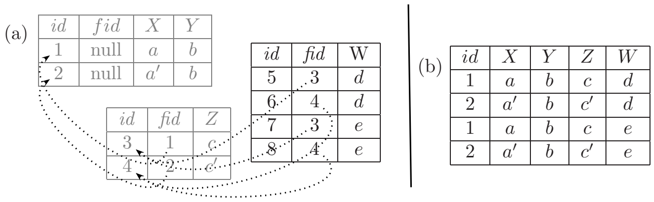
(b)

id	*fid*	*Z*
3	1	c
4	2	c'

Fig. 1. (a) Initial table t_0. (b) Table for pair $\langle \hat{h}_1 = (h(X, Y) \leftarrow q^{bf}(X, Z).), t_1 \rangle$.

Continuing this example, let the facts $r(b, d)$ and $r(b, e)$ be in the base table for background predicate r. We represent the hypothesis $\hat{h}_2 = (h(X, Y) \leftarrow q^{bf}(X, Z), r^{bf}(Y, W).)$ as pair $\langle \hat{h}_2, t_2 \rangle$, where t_2 is shown in Fig. 2(a) along with its references to t_1 for illustration.

We could avoid indirection and store all variable bindings for a corresponding seed in each row. However, significant savings are obtained by not storing shared bindings redundantly. For example, given pair $\langle \hat{h}_1, t_1 \rangle$ above, all hypotheses that are extensions of $\hat{h}_1$, including $\hat{h}_2$, share bindings for variables appearing in table t_1. Using chained tables allows these bindings to be stored once, and extensions then refer to these bindings through indirection. Otherwise, each new table would store a copy of table t_1 along with any new bindings. The use of unique row ids allows tables to be unambiguously reconstructed from the chained tables when necessary. For the running example, the reconstructed table for t_2 is shown in Fig. 2(b).



(a)

id	*fid*	*X*	*Y*
1	null	a	b
2	null	a'	b

id	*fid*	*Z*
3	1	c
4	2	c'

id	*fid*	W
5	3	d
6	4	d
7	3	e
8	4	e

(b)

id	*X*	*Y*	*Z*	*W*
1	a	b	c	d
2	a'	b	c'	d
1	a	b	c	e
2	a'	b	c'	e

Fig. 2. (a) Table t_2 in pair $\langle \hat{h}_2, t_2 \rangle$ with its references to t_0 and t_1. (b) Reconstructed version of t_2.

3.1 Extension-Join

Using this representation, we formulate hypothesis extension as a stylized join operation on database tables [14]. This operation takes as input two intensional/extensional pairs as described above. The first $\langle \hat{h}_n = (h \leftarrow r_1, \ldots, r_n), t_n \rangle$ is a hypothesis pair as in definition 4. The second, $\langle p, \text{pred}(p) \rangle$, consists of a moded literal p and its corresponding base table pred(p). The result of the operation is a new hypothesis pair $\langle \hat{h}_{i+1} = (h \leftarrow r_1, \ldots, r_n, p), t_{i+1} \rangle$. Each substitution in t_i is extended according to the moded literal and those that make the new hypothesis h_{i+1} cover the corresponding seed $e \in E$ are retained in the new table t_{i+1}. We denote this operation as $\langle \hat{h}_{i+1}, t_{i+1} \rangle = \langle \hat{h}_i, t_i \rangle \bowtie \langle p, \text{pred}(p) \rangle$.

The extension-join operation combines a number of steps, the most significant of which is an equi-join of the input tables. The remaining steps are for bookkeeping, and set up the equi-join to capture the proper variable bindings of the extension. Extension-join is defined by Algorithm 1.

```
Algorithm 1: The Extension-Join operation
Input: Hypothesis pair $\langle h_i, t_i \rangle$ and extension $\langle p, \text{pred}(p) \rangle$
Output: Extended hypothesis pair $\langle h_{i+1}, t_{i+1} \rangle$
X-JOIN($\langle h_i, t_i \rangle, \langle p, \text{pred}(p) \rangle$)
(1)    compute projection of $t_i$
(2)    build join constraints and result projection list
(3)    execute join and result projection
(4)    if result is not empty make new table $t_{i+1}$
(5)    let $h_i = (h \leftarrow r_1, \ldots, r_n.)$, and set $h_{i+1} = (h \leftarrow r_1, \ldots, r_n, p.)$
(6)    output $\langle h_{i+1}, t_{i+1} \rangle$
```

We present the details of these steps using the running example. To create pair $\langle \hat{h}_2, t_2 \rangle$ we calculate $\langle \hat{h}_1, t_1 \rangle \bowtie \langle r^{bf}(Y, W), \text{pred}(r) \rangle$, where t_1 is shown in Fig. 1(b) and the table pred(r) contains the facts $r(b, d)$ and $r(b, e)$.

1. Compute projection of t_i. We project the input hypothesis table to only those columns containing the bound arguments of the extension, following *fid* fields to gather necessary variable bindings. We also keep the id and fid columns required for chaining rows in the result t_{i+1} to rows in t_i. In our example, t_1 is projected to table t'_1 shown in Fig. 3(a). For extension $r^{bf}(Y, W)$ only column Y is needed since Y is the only *bound* argument of $r^{bf}(Y, W)$.

(a)

id	fid	Y
3	1	b
4	2	b

(b)

$t.id$	$t.fid$	pred(r).2
3	1	d
4	2	d
3	1	e
4	2	e

Fig. 3. (a) Table t'_1, the projection of t_1 to bound arguments of extension. (b) Result of equi-join of t'_1 and pred(r) after projection to identifier and free argument columns.

In principle, several extensions to a given hypothesis require the same bound variables from the input hypothesis table. The result of extension-joining each of these extensions and the input hypothesis can be computed simultaneously using a single projection of the input hypothesis table. This first step in the extension-join operation permits set-oriented optimizations of this kind.

2. Build join constraints and result projection list . This step finds common bindings between the input hypothesis and its extension using the input moded literal. These bindings are expressed as constraints on an equi-join operation, the result of which is then projected to only those columns required for chaining and those containing new variables.

For a given pair of operands, a list c of join constraints of the form $t_i.j = p.k$ is constructed, where j is a variable column in t_i and k is a column of base table p. For our current example, $c = \{t'_1.Y = \text{pred}(r).1\}$ since the first column of pred(r) is specified as a *bound* argument and variable Y is assigned to that column in $r^{bf}(Y, W)$. A list of column names l is constructed as $\{t_i.id, t_i, fid, p.x_1, \ldots, p.x_m\}$ where $p.x_1, \ldots, p.x_m$ are the columns of base table p that do not appear in the join constraints in list c. In the example $l = \{t'_1.id, t'_1.fid, \text{pred}(r).2\}$ since column 2 of pred(r) is not involved in any constraint in list c.

3. Execute join and projection . The result of the previous step is used to execute an equi-join on the two input tables. This operation is defined by the relational algebra [14] expression $\pi_l(t_i \bowtie_c p)$ where $\bowtie_c$ is an equi-join under the constraints given in list c, and π_l is a projection to the columns listed in l. This has the effect of extending substitutions in input table t_i with bindings from the input base predicate. For our running example, the result of $\pi_{\{t'_1.id, t'_1.fid, \text{pred}(r).2\}}(t'_1 \bowtie_{t'_1.Y=\text{pred}(r).1} \text{pred}(r))$ is shown in Fig. 3(b).

4. If result is not empty, build new table t_{i+1} . This step transforms the result of the previous step so it conforms to the hypothesis schema. It also chains the rows in t_{i+1} to rows of t_i by making the proper entries in the *fid* column of the new table. Column names for the new table are derived from the moded literal and a unique id is generated for each row in the result. The final result for our example, t_2, was shown in Fig. 2.

By Definition 4 and the extension-join in Algorithm 1, all hypothesis tables contain a unique identifier for each row, and refer to the unique identifier of a parent row. Since the seed table t_0 contains exactly one row for each seed example e in seed set E, a row identifier e_{id} can be uniquely associated with each seed example. Thus, any row in subsequent tables can be associated with a seed example $e \in E$ using the row identifier e_{id} in seed table t_0, by following *fid* links. We define a selection operation, denoted $\sigma_e(t)$, and a projection operation, denoted $E(t)$, that use these row identifiers:

Definition 5. *Let* $\hat{h} = (h \leftarrow r_1, \ldots, r_n)$ *and*

$$\langle \hat{h}, t \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle$$

such that seed table t_0 is built from a seed example set E, and t_x is the reconstruction of t through fid *fields as described above. Then:*

1. $E(t) \overset{def}{=} \pi_{id}(t_x)$ *is the projection of t to its example identifiers, where π is the relational algebra projection operator.*
2. $\sigma_e(t) \overset{def}{=} \sigma_{id=e_{id}}(t_x)$ *is the selection of t to rows involving seed e, where e_{id} is the row identifier for seed $e \in E$ in the initial table t_0, and σ is the relational algebra selection operator.*

A useful property of the extension-join operation is that selection on examples for a hypothesis table can be pushed to a selection on the original table of seeds t_0. We formalize this with the following lemma:

Lemma 1. *Let* $\hat{h}_m = (h \leftarrow r_1, \ldots, r_m.) \in H(\mathcal{M}, h)$,

$$\langle \hat{h}_m, t_m \rangle = \langle (h \leftarrow \mathrm{true.}), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_m, \mathrm{pred}(r_m) \rangle,$$

such that seed table t_0 is built from a seed example set E, and let $\langle \hat{h}_n, t_n \rangle = \langle \hat{h}_m, t_m \rangle \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle$.

For every $e \in E$, if $\langle \hat{h}_n, t_e \rangle \overset{def}{=} \langle \hat{h}_m, \sigma_e(t_m) \rangle \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle$, *then* $t_e = \sigma_e(t_n)$, *where σ_e is the selection operation of Definition 5.*

Proof. Let e_{id} be the unique row identifier for example e in seed table t_0.

($\sigma_e(t_n) \subseteq t_e$). Let $t_e \subset \sigma_e(t_n)$, then there is a tuple $s \in \sigma_e(t_n)$ such that $s \notin t_e$. Let s be the result of joining tuples $s' \in t_m$ and $s'' \in \mathrm{pred}(r_n)$ according to the definition of extension-join. Since $s \in \sigma_e(t_n)$, the $s.id = e_{id}$, by definition of extension-join, $s'.id = s.id. = e_{id}$. However, since $s \notin t_e$, the definition of extension-join implies $s' \notin \sigma_e(t_m)$. This is a contradiction since we established $s'.id = e_{id}$.

($t_e \subseteq \sigma_e(t)$). Conversely, let $\sigma_e(t_n) \subset t_e$, then there is a tuple $s \in t_e$ such that $s \notin \sigma_e(t_n)$. Let s be the result of joining tuples $s' \in \sigma_e(t_m)$ and $s'' \in$

pred(r_n) according to the definition of extension-join. Since $s' \in \sigma_e(t_m)$, $s' \in t_m$ which implies by definition of extension-join that $s \in t_n$. Furthermore, since $s' \in \sigma_e(t_m)$, we have $s'.id = e_{id}$, and the definition of extension-join implies $s.id = e_{id}$. We have shown that $s \in t_n$ and $s.id = e_{id}$, but $s \notin \sigma_e(t_n)$. This is a contradiction. □

It is worth noting that existing work addresses issues we present here. For example, data structures used in algorithms for testing θ-subsumption [15–17] store multiple substitutions compactly to avoid backtracking when finding satisfying substitutions. However, the compact representation used can make maintaining statistics of the type we discuss below difficult. On the other hand, the data structure for storing multiple substitutions used in the LogAn-H system [18] uses the reconstructed tables we discuss above which store information redundantly.

Techniques that store only coverage lists or some computed answers meet some, but not all, of our goals. For example, storing coverage lists [19] or a technique such as tuple-id propagation [20] allows for compact storage and fast retrieval of statistics used to determine coverage measures of a hypothesis in a classification setting. However other types statistics, those not involving coverage as used in some probabilistic models, for example, are not easily derivable. We present a formulation that seeks to balance the two goals of caching and availability of general statistics.

3.2 Benefits of the WILD Representation

We identify two general areas in which our representation offers advantages:

1. Within the current search and score paradigm in ILP, this framework allows for efficiency, scalability and flexibility.
2. This framework easily adapts to settings where learning theories in languages other than Horn clauses is desired.

We discuss these benefits below.

Caching Benefits. Each table contains those bindings required to determine the coverage properties of a hypothesis $\hat{h}$ with respect to the seed table t_0. Once these bindings are cached by pair $\langle \hat{h}, t \rangle$, they can be reused to determine coverage properties of extensions to $\hat{h}$. This is exploited in the context of search-space restriction in the next section.

Set-oriented Hypothesis Extension. The extension-join operation can be carried out efficiently in a relational database system since it is defined in terms of relational operations. Thus, ILP could potentially be carried out on disk-resident data.

There are also set-oriented optimizations that can be performed at the tuple level during extension-join. For instance, earlier we described an optimization where extension-join on a particular hypothesis and a set of its extensions (for example, modes that share a base table) is executed in a set-oriented fashion. This optimization is in the spirit of the query packs presented by Blockeel et al. [5]

Alternative Search Methods The cached table for a hypothesis pair can be used to maintain statistics that help in defining the hypothesis search-space, and in exploring that space. In this paper we present an application in which statistics from extensional tables are used to restrict the space of allowable hypotheses.

As another example, stochastic search methods can use prior distributions over the space to guide search towards probably useful parts of the space [7, 8]. A hypothesis space generated and, thus, defined using our representation can use an informative prior derived from coverage statistics derivable from cached tables.

Similarly, estimates of a given property of hypotheses can also be used to guide search. For example, estimates of the coverage of a hypothesis may be used to specify which parts of the hypothesis space to explore [4, 9]. Under our representation, these estimates are derived from a cached table resulting from the extension-join of background predicates to some representative set of seeds. Another method might use our representation to estimate how efficiently a hypotheses can be evaluated. For example, Struyf and Blockeel [3] estimate a prior on the selectivity of literals to decide an efficient literal reordering for a given hypothesis. Statistics derived from our representation can provide good estimates of the selectivity of a literal.

Language Extensions This framework permits learning rules in languages other than sets of Horn clauses. For example, we can use statistics in the cached table for a hypothesis to train a statistical model that infers missing values in other instances of similar datasets. This is the formulation for CLP(BN) [11], a language easily incorporated into our hypothesis framework.

Alternatively, we can use statistics in the cached table to make distributional statements regarding variables in a hypothesis. For example, we can estimate the distribution of a column in our target predicate and determine its correlation to subsets of other columns in the background knowledge using statistics derived from cached tables. This allows for statements of the type *rich people tend to live in big houses* to be made in the learned program.

Extensions to the Datalog language have been proposed that add the ability to group constants and calculate aggregates on these groups [21]. By having cached tables available, these groups can be defined and aggregates calculated on the fly during the learning process. This may allow for statements about aggregates like those described by Vens et al. [10]. Another Datalog extension is the use of negated literals in clauses. This is allowed through the requirement that programs be stratified with respect to negation. Since we assume a set-oriented, bottom-up evaluation strategy in our system, we can expect to learn stratified programs with negated literals.

The remainder of this paper presents an initial example of the benefits of the WILD representation and the avenues it opens when defining the ILP task. We look at how the space of allowable hypotheses can be defined using background knowledge about multiple facts in the target predicate.

4 An Initial Application

The Aleph and Progol systems restrict the set of hypotheses in the search space through *Inverse Entailment* [13]. A specific seed example is chosen to generate a set of literals, known as the bottom clause, by finding facts in the background knowledge that are relevant to the chosen seed example. The space of hypotheses is restricted to include only generalizations of this bottom clause, consequently, all hypotheses generated will cover the seed example. This process seeks to restrict search to useful hypotheses. However, a seed defines a space of hypotheses that are useful only in respect to that seed.

In the presence of noise in data, restricting the search space based on a single seed is potentially wasteful. For instance, suppose that a few positive examples are mislabelled, and are in fact negative examples. Using any of these examples as seeds will restrict the search space to hypotheses that probably cover many negative examples. Hypothesis evaluation based on coverage will then try to find a, possibly non-existent, very specific clause that differentiates between negative examples. While search parameters can be used to alleviate this, a minimum positive example coverage constraint for example, a principled method that avoids this phenomenon while defining the search space is best.

If a ‘usefulness’ restriction is imposed on the search space defined in terms of multiple seeds, then the effect of an unfavorable choice of seed might be mitigated by true representative seeds in the set. Furthermore, it would be useful to provide a degree of freedom in how the space restriction is defined in terms of the multiple seeds. We show how a hypothesis generation strategy using the WILD representation meets these goals by generalizing Inverse Entailment to multiple seeds.

First, we describe Inverse Entailment in more detail and then show how the WILD representation is used to define a hypothesis space. Specifically, how it is used to define a space using Inverse Entailment in the single-seed case. Finally, we define a generalization of Inverse Entailment and show it defines a class of ‘usefulness’ restrictions that can be imposed on the hypothesis space.

4.1 Inverse Entailment

Inverse Entailment constructs a set of literals that defines the allowable hypotheses in the search space. In practice, this construction is done using a partitioning approach: the first partition contains the constants appearing in the seed example; at each step of the iteration, instances of the constants in the current partition are found in each background predicate as specified in any of the binding patterns defined by the given modes; each ground literal containing instances is added to the bottom clause and new constants appearing in these literals are added to the next partition if the corresponding argument is specified as *free* in some mode. This is repeated until no new constants are added to the next partition or a user-defined bound on the number of iterations performed is met. To finalize, the ground literals in the bottom clause are ‘variabilized’ according to the given set of modes. Here and below we leave out a bound on hypothesis length for clarity, but this can be easily implemented.

Allowable hypotheses in the search space are valid ordered subsets of the literals in the bottom clause. Literals can only be used to extend a hypothesis if it appears in the bottom clause and variables appearing in its *bound* arguments must appear in the hypothesis to be extended. The subset of $H(\mathcal{M}, h)$ built from a given bottom clause can now be defined.

Definition 6. *Given a set of allowable moded literals $\mathcal{M}$, target predicate h, seed example e, depth bound k and background knowledge $\mathcal{B}$, let $\bot_e$ be the bottom clause built from seed example e. Define the set of hypotheses $H_A(e)$ generated from $\bot_e$ as $H_A(\mathcal{M}, h, e, k, \mathcal{B}) = \{(h \leftarrow \text{true}.)\} \cup \{h \leftarrow r_1, \ldots, r_n, p. : (h \leftarrow r_1, \ldots, r_n.) \in H_A(\mathcal{M}, h, e, k, \mathcal{B}), p \in \bot_e,$ and $\text{bound}(p) \subseteq \text{Vars}(h \leftarrow r_1, \ldots, r_n.).\}$*

Remark 1. We stated previously that all hypotheses in set $H_A(\mathcal{M}, h, e, k, \mathcal{B})$ will cover the seed example e. Muggleton proved in [13] that Inverse Entailment is complete under θ-subsumption, thus if the depth bound is relaxed, that is, if $k = \infty$, for a given $\hat{h} \in H(\mathcal{M}, h)$, $\hat{h} \wedge \mathcal{B} \vdash e$ if, and only if, $\hat{h} \in H_A(\mathcal{M}, h, e, \infty, \mathcal{B})$.

4.2 WILD Hypothesis Generation

This process of Inverse Entailment can be generalized to a set-oriented formulation. Instead of a single seed example being used to restrict the search space, a set of examples is used along with a filter function that determines which candidate hypotheses can be included in the search space. This generalized version of Inverse Entailment, like the original Aleph/Progol version, benefits from bottom-up computation. Using the representation and the extension-join operation of Section 3, we propose the following algorithm for generating hypotheses:

Algorithm 2: WILD Hypothesis Generator

Input: Set of allowable moded literals $\mathcal{M}$, target predicate h, seed fact set E, depth bound k, background knowledge $\mathcal{B}$, filter-function ϕ

Output: Set of hypotheses

```
GenerateH($\mathcal{M}, h, E, k, \mathcal{B}, \phi$)
    openset = $\{\langle (h \leftarrow \text{true}.), t_0 \rangle\}$, ($t_0$ built from seed set $E$)
    output $\langle (h \leftarrow \text{true}.), t_0 \rangle$
    while openset is not empty
        choose and remove $\langle \hat{h}_i, t_i \rangle$ from openset
        foreach moded predicate $p$ that is a valid extension to $\hat{h}_i$
            compute $\langle \hat{h}_{i+1}, t_{i+1} \rangle = \langle \hat{h}_i, t_i \rangle \bowtie \langle p, \text{pred}(p) \rangle$
            if $\phi(t_{i+1})$ is true
                output $\langle \hat{h}_{i+1}, t_{i+1} \rangle$ as an allowable hypothesis
                if depth of $\hat{h}_{i+1} \leq k$
                    add $\langle \hat{h}_{i+1}, t_{i+1} \rangle$ to openset
```

Valid extensions here are as in set $H(\mathcal{M}, h)$, that is: $\text{bound}(p) \subseteq \text{Vars}(h_i)$ so that arguments marked *bound* are assigned a variable already appearing in the hypothesis to be extended. In step 9, the depth of h_{i+1} as defined by Muggleton in [13] is easily obtained from t_{i+1}. First, we define $\hat{h}_0 = (h \leftarrow \text{true})$ to have depth 0. If the depth of $\hat{h}_i$ is j, and table t_{i+1} has columns for variables not appearing in t_i, then the depth $\hat{h}_{i+1}$ is $j+1$.

Using the WILD hypothesis generator we formulate a strategy to enumerate the set of hypotheses generated by Inverse Entailment in the single-seed case. First, we make the seed set $E = \{e\}$ a singleton set. We then set the filter function ϕ to take a table as input, and return true if the table is not empty. We'll denote this emptiness-testing function as ϕ_{empty}. We prove in the following proposition that with these parameters, we can generate a complete and sound set of hypotheses that cover the given single seed e.

Proposition 1. ***Soundness and Completeness of WILD Generation for a Single Seed.*** *Let $H_B(e) = \textsc{GenerateH}(\mathcal{M}, h, \{e\}, \infty, \mathcal{B}, \phi_{\text{empty}})$. Hypothesis $\hat{h} \in H(\mathcal{M}, h)$ covers e if, and only if, there exists table t such that $\langle \hat{h}, t\rangle \in H_B(e)$.*

Proof. (*Only if*). Proceeds by induction on n, the number of literals in $\hat{h}$. If $n = 1$ we have that $\hat{h}$ must be of the form $(h \leftarrow \text{true.})$. $\hat{h}$ covers e by a unique substitution θ that maps variables in h to constants in e such that $h[\theta] = e$. That is, the result of applying substitution θ to literal h is e. By construction, in Step 2 of Algorithm 2, we have $\langle \hat{h}, t_0\rangle \in H_B(e)$.

Let the 'only if' direction of the Proposition be true for all $n \leq m-1$; thus we assume that for each $\hat{h}_{m-1} = (h \leftarrow r_1, \ldots, r_{m-1}.) \in H$ that covers e, there exists a table t_{m-1} such that $\langle \hat{h}_{m-1}, t_{m-1}\rangle \in H_B(e)$. We show that for each extension $\hat{h}_m$ to $\hat{h}_{m-1}$ such that $\hat{h}_m = (h \leftarrow r_1, \ldots, r_{m-1}, r_m.)$ covers e, there is a pair $\langle \hat{h}_m, t_m\rangle \in H_B(e)$.

Since $\hat{h}_m$ covers e we must have that $\hat{h}_{m-1}$ also covers e, and thus by the inductive hypothesis, there is table t_{m-1} such that $\langle \hat{h}_{m-1}, t_{m-1}\rangle \in H_B(e)$. Also, since $\hat{h}_m$ covers e, there exists a substitution θ such that $h[\theta] = e$ and for each i, $1 \leq i \leq m$ there is a tuple $s \in \text{pred}(r_i)$ such that $r_i[\theta] = s_i$, specifically there is a tuple $s_m \in \text{pred}(r_m)$ such that $r_m[\theta] = s_m$. Let

$$\langle \hat{h}_m, t_m\rangle = \langle \hat{h}_{m-1}, t_{m-1}\rangle \bowtie \langle r_m, \text{pred}(r_m)\rangle,$$

then due to tuple $s_m \in \text{pred}(r_m)$ and the definition of extension-join, there is at least one tuple in t_m, that is, $\phi_{\text{empty}}(t_m) = \text{true}$. This implies $\langle \hat{h}_m, t_m\rangle \in H_B(e)$ as desired.

(*If*). Now we prove that if there is a pair $\langle \hat{h}, t\rangle \in H_B(e)$, then $\hat{h}$ covers e. Proceed by induction on n, the number of literals in $\hat{h}$. If $n = 1$ then we have $\langle \hat{h} = (h \leftarrow \text{true.}), t_0\rangle \in H_B(e)$, and by construction we have that t_0 is built from the constants appearing in e. We build a substitution θ from t_0 such that $h[\theta] = e$ which makes $\hat{h}$ cover e.

Let the 'if' direction of the claim be true for all $n \leq m-1$, and assume that if $\langle \hat{h}_{m-1} = (h \leftarrow r_1, \ldots, r_{m-1}.), t_{m-1}\rangle \in H_B(e)$ then $\hat{h}_{m-1}$ covers e. We show that for each extension $\langle \hat{h}_m, t_m\rangle = \langle \hat{h}_{m-1}, t_{m-1}\rangle \bowtie \langle r_m, \text{pred}(r_m)\rangle \in H_B(e)$, $\hat{h}_m = (h \leftarrow r_1, \ldots, r_{m-1}, r_m.)$ covers e.

Since $\langle \hat{h}_m, t_m \rangle \in H_B(e)$, we have that $\langle \hat{h}_{m-1}, t_{m-1} \rangle \in H_B(e)$. By the inductive hypothesis, there is a substitution θ' such that $h[\theta'] = e$ and there is a tuple $s_i \in \text{pred}(r_i)$ for all $i,\ 1 \leq i \leq m-1$, such that $r_i[\theta'] = s_i$. Since $\langle \hat{h}_m, t_m \rangle \in H_B(e)$ we know that $\phi_{\text{empty}}(t_m) = \text{true}$, thus there is at least one tuple $s_m \in t_m$. We build a substitution σ with a domain consisting of variables not appearing in θ', which corresponds to arguments of r_m in free(r_m). We bind the variables in σ to the constants appearing in the corresponding arguments in tuple s_m. Since bound(r_m) is a subset of the domain of θ', we can build the substitution $\theta = \theta'\sigma$. The result is that $h[\theta] = e$, and there is a tuple $s_i \in \text{pred}(r_i)$ for all $i,\ 1 \leq i \leq m-1$ such that $r_i[\theta] = s_i$. Finally due to the definition of equi-join, there is a tuple $s_m \in \text{pred}(r_m)$ such that $r_m[\theta] = s_m$. This implies $\hat{h}_m$ covers e as desired. □

The following corollary follows from Proposition 1 and Remark 1.

Corollary 1. ***WILD Generates the Inverse Entailment Space for a Single Seed.*** *Let $H_B(e) = \text{GenerateH}(\mathcal{M}, h, \{e\}, \infty, \mathcal{B}, \phi_{\text{empty}})$, and $H_A(e)$ be as in Definition 6, then for every $\hat{h} \in H(\mathcal{M}, h)$, $\hat{h} \in H_A(e)$ if, and only if there exits table t such that $\langle \hat{h}, t \rangle \in H_B(e)$.*

Proof. Follows trivially from Proposition 1 and Remark 1. as both sets contain exactly the subset of $H(\mathcal{M}, h)$ that cover single seed e. □

5 Generalized Inverse Entailment

We now present a scheme for generalizing Inverse Entailment using multiple seeds. It uses the parameters in the WILD generation algorithm to restrict the set of allowable hypotheses. Specifically, the filter function is used to only allow generation of hypotheses that meet some coverage criteria. Inverse Entailment is generalized in the sense that while the criterion used for restriction in Inverse Entailment is that hypotheses cover a single seed, we use a class of measures of the coverage of a hypotheses over the set of seed examples. This class of measures is implemented as the filter function ϕ of the WILD generation algorithm.

Given a set of seed examples we denote the subset of seeds covered by a hypothesis $\hat{h} \in H(\mathcal{M}, h)$ as $E(\hat{h})$.

Definition 7. *Let $\hat{h} \in H(\mathcal{M}, h)$, then: $E(\hat{h}) = \{e \in E \mid \hat{h} \in H_A(e)\}$, where $H_A(e)$ is the set of hypotheses generated by Inverse Entailment from the single seed example e as in Definition 6.*

Intuitively, $e \in E(\hat{h})$ if $\hat{h}$ is a hypothesis generated from the bottom clause built from seed e. This is equivalent to stating, due to Remark 1, that $e \in E(\hat{h})$ if $\hat{h}$ covers e.

We define two filter functions we propose and evaluate in this paper. Given a subset $E' \subseteq E$, define:

1. *Intersection:* $\phi_{\text{int}}(E') = \text{true}$ if $E' = E$. We will claim in Proposition 2 that the hypotheses generated by the WILD hypothesis generator using this filter are those hypotheses that would be generated by *every* bottom clause built, in turn, by a seed in E. That is, the resulting space is the intersection of the spaces defined by the set of Aleph bottom clauses.
2. *Support:* $\phi_{\text{sup}}(E') = \text{true}$ if, for a given threshold η, $|E'|/|E| \geq \eta$. We borrow this concept from frequent itemset and relational pattern mining algorithms, [22, 23] and use it to generalize the coverage assumptions of Inverse Entailment. This introduces an extra parameter that can be used to determine the amount of filtering to apply. We will claim in Proposition 2 that the hypotheses generated using this filter are those that cover at least $\eta|E|$ seed examples. Notice that if $\eta = 1$, this is equivalent to ϕ_{int} above.

Notice that these functions are monotonic on the size of seed subset E' and that they return true for the entire set E. We formalize this in the following definition and use these properties when proving our main result in Proposition 2.

Definition 8. ***Proper Filter Function*** *Let E be a set of seed examples and $E' \subseteq E$. A filter function ϕ is* ***proper*** *when (1) if $\phi(E') = \text{true}$ then for any superset $E'' \supseteq E'$: $E'' = \text{true}$; and (2) $\phi(E) = \text{true}$.*

Below, we take $\phi(t)$ where t is a table in an intensional/extensional pair, to mean $\phi(E(t))$ where $E(t)$ is the projection of t to its example identifiers as in Definition 5.

We can now present our main result regarding generalized Inverse Entailment which states that the WILD generator will produce only, and all, hypotheses in $H(\mathcal{M}, h)$ that meet the criteria imposed by the filter function. That is, it generates those hypotheses we deem as useful with respect to the set of seed examples.

Proposition 2. ***Soundness and Completeness of WILD Generation for Multiple Seeds.*** *Let $H_W(E) \overset{def}{=} \textsc{GenerateH}(\mathcal{M}, h, E, \infty, \mathcal{B}, \phi)$ be the set of hypotheses generated by the WILD generator for seed set E such that ϕ is a proper filter as specified in Definition 8. For all $\hat{h} \in H(\mathcal{M}, h)$, $\phi(E(\hat{h})) = \text{true}$ if, and only if, there exists a table t such that $\langle \hat{h}, t\rangle \in H_W(E)$.*

We use two Lemmas to prove this result. Once these are stated and proven we present the proof of Proposition 2. Throughout we denote as $H_B(e) \overset{\text{def}}{=} \textsc{GenerateH}(\mathcal{M}, h, \{e\}, \infty, \mathcal{B}, \phi_{\text{empty}})$ the set of hypotheses described in Proposition 1, that is, the set of hypotheses generated by the WILD hypothesis generator for single seed e, using the emptiness-testing function ϕ_{empty}. Also, we denote as $H_A(e)$ the set from Definition 6, that is, the set of hypotheses generated by Inverse Entailment using single seed e. Recall from Corollary 1 that sets $H_A(e)$ and $H_B(e)$ are equal.

First, we use the selection result in Lemma 1 to reason about pairs $\langle \hat{h}, t\rangle$, built from multiple seeds, in terms of the single-seed space $H_B(e)$. We state in the next Lemma the conditions in which the projection of t to its seed row identifiers contains the identifier for a particular example e. We show this occurs if, and only if, the pair $\langle \hat{h}, \sigma_e(t) \rangle$ is in the single-seed space $H_B(e)$, or equivalently, due to Proposition 1, when $\hat{h}$ covers e. Here $\sigma_e(t)$ is the selection of rows in t involving seed e as defined in Definition 5

Lemma 2. *Let $\hat{h}_n = (h \leftarrow r_1, \ldots, r_n.) \in H(\mathcal{M}, h)$, and*

$$\langle \hat{h}_n, t_n \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle,$$

such that seed table t_0 is built from a seed example set E as described in Section 3. Let e_{id} be the unique row identifier associated with seed example $e \in E$ in table t_0. For every $e \in E$, $e_{id} \in E(t)$ if, and only if, $\langle \hat{h}, \sigma_e(t) \rangle \in H_B(e)$.

Proof. (If). If $\langle \hat{h}, \sigma_e(t) \rangle \in H_B(e)$, by definition of ϕ_{empty}, $\sigma_e(t_n) \neq \emptyset$. This implies $e_{id} \in E(t)$.

(Only if). We proceed by induction on n, the number of literals in the body of $\hat{h}$. If $n = 1$, then by construction $\langle \hat{h}, \sigma_e(t) \rangle \in H_B(e)$.

Let the 'only if' direction of the Lemma be true for all $n \leq m - 1$; we assume that for every

$$\langle \hat{h}_{m-1}, t_{m-1} \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_{m-1}, \mathrm{pred}(r_{m-1}) \rangle,$$

such that seed table t_0 is built from a seed example set E as described in Section 3, and $e \in E$, $e_{id} \in E(t_{m-1})$ implies $\langle \hat{h}_{m-1}, \sigma_e(t_{m-1}) \rangle \in H_B(e)$.

Let $\langle \hat{h}_m, t_m \rangle = \langle \hat{h}_{m-1}, t_{m-1} \rangle \bowtie \langle r_m, \mathrm{pred}(r_m) \rangle$, and $e_{id} \in E(t_m)$, we now show $\langle \hat{h}, \sigma_e(t_m) \rangle \in H_B(e)$. Since $e_{id} \in E(t_m)$ the selection $\sigma_e(t_m) \neq \emptyset$. By the definition of extension-join and σ_e, this implies that $\sigma_e(t_{m-1}) \neq \emptyset$, and thus $e_{id} \in E(t_{m-1})$. By the inductive hypothesis pair $\langle \hat{h}_{m-1}, \sigma_e(t_{m-1}) \rangle \in H_B(e)$. Therefore, since $\langle \hat{h}_{m-1}, \sigma_e(t_{m-1}) \rangle \in H_B(e)$, and $\sigma_e(t_m) \neq \emptyset$, $\langle \hat{h}_m, \sigma_e(t_m) \rangle \in H_B(e)$ as desired. □

The result of $E(t)$ is a set of row identifiers for examples in seed set E. Given this set of identifiers a subset of examples in seed set E can be uniquely specified. The next Lemma shows that under some conditions, for a hypothesis pair $\langle \hat{h}, t \rangle$ the set of example identifiers in t can be mapped to the set of examples covered by $\hat{h}$. We denote this relationship as $E(t) = E(\hat{h})$. This mapping will lead directly to the desired result for Proposition 2, except for the issues presented by the filter function ϕ. Addressing those effects will be the bulk of the proof of Proposition 2.

Lemma 3. *Let $\hat{h} = (h \leftarrow r_1, \ldots, r_n.) \in H(\mathcal{M}, h)$, and*

$$\langle \hat{h}, t \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle,$$

such that seed table t_0 is built from a seed example set E as described in Section 3. Let e_{id} be the unique row identifier associated with seed example $e \in E$ in table t_0. Then for every $e \in E$, $e_{id} \in E(t)$, if and only if, $e \in E(\hat{h})$, where $E(t)$ is the projection of table t as in Definition 5 and $E(\hat{h})$ is the set of examples covered by hypothesis $\hat{h}$ as in Definition 7. That is, $E(t) = E(\hat{h})$.

Proof. (*If*). By Definition 7, $e \in E(\hat{h})$ if $\hat{h} \in H_A(e)$. By Proposition 1, if $\hat{h} \in H_A(e)$ then there exists table t_e such that $\langle \hat{h}, t_e \rangle \in H_B(e)$. This implies by Lemma 1 that $e_{id} \in E(t)$ the projection of t to its seed example identifiers.

(*Only if*). By Lemma 1, $e_{id} \in E(t)$ implies $\langle \hat{h}, \sigma_e(t) \rangle \in H_B(e)$. By Proposition 1, $\langle \hat{h}, t_e \rangle \in H_B(e)$ implies $\hat{h} \in H_A(e)$. Then by definition, $e \in E(\hat{h})$, that is, $\hat{h}$ covers single seed e. □

We now prove Proposition 2 proceeding by induction on the number of literals in the hypothesis. We note that if a hypothesis $\hat{h}_n \in H(\mathcal{M}, h)$ is an extension of hypothesis $\hat{h}_{n-1} \in H(\mathcal{M}, h)$, the set of examples covered by $\hat{h}_n$ is a subset of the examples covered by $\hat{h}_{n-1}$. We use the monotonicity of the *proper* filter function ϕ to reason about the result of applying ϕ to the set of examples covered by $\hat{h}_n$. Finally, Lemmas 2 and 3 provide a mapping from the set of examples covered by a hypothesis to the examples present in a table resulting from a chain of multiple extension-joins starting from the seed table.

Proof. *(If).* Let $\hat{h} = (h \leftarrow r_1, \ldots, r_n.) \in H(\mathcal{M}, h)$. If $\langle \hat{h}, t \rangle \in H_W(E)$, then

$$\langle \hat{h}, t \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_n, \mathrm{pred}(r_n) \rangle,$$

such that seed table t_0 is built from a seed example set E as described in Section 3 and $\phi(t) \overset{\text{def}}{=} \phi(E(t)) = \text{true}$. We have by Lemma 3 that $E(\hat{h}) = E(t)$, and thus, $\phi(E(\hat{h})) = \text{true}$ since $\phi(E(t)) = \text{true}$.

(Only if). We proceed by induction on n, the number of literals in the body of $\hat{h}$. If $n = 1$, then $\hat{h} = (h \leftarrow \text{true}.)$. By construction there is a t_0 such that $\langle \hat{h}_n, t_0 \rangle \in H_W(E)$.

Let the 'only if' direction of the Proposition be true for all $n \leq m - 1$; we assume that for every $\hat{h}_{m-1} = (h \leftarrow r_1, \ldots, r_{m-1}.) \in H(\mathcal{M}, h)$, $\phi(E(\hat{h}_{m-1})) = \text{true}$ implies that there is a table t_{m-1} such that $\langle \hat{h}_{m-1}, t_{m-1} \rangle \in H_W(E)$. Let $\hat{h}_m = (h \leftarrow r_1, \ldots, r_{m-1}, r_m) \in H(\mathcal{M}, h)$ and $\phi(E(\hat{h}_m)) = \text{true}$. We show there is a table t_m such that $\langle \hat{h}_m, t_m \rangle \in H_W(E)$.

By the monotonicity of ϕ, $\phi(E(\hat{h}_{m-1})) = \text{true}$ since $E(\hat{h}_m) \subseteq E(\hat{h}_{m-1})$ and $\phi(E(\hat{h}_m)) = \text{true}$. Then, by the inductive hypothesis, there exists $\langle \hat{h}_{m-1}, t_{m-1} \rangle \in H_W(E)$ where

$$\langle \hat{h}_{m-1}, t_{m-1} \rangle = \langle (h \leftarrow .), t_0 \rangle \bowtie \langle r_1, \mathrm{pred}(r_1) \rangle \bowtie \cdots \bowtie \langle r_{m-1}, \mathrm{pred}(r_{m-1}) \rangle,$$

such that table t_0 is built from a seed example set E as described in Section 3 and $\phi(t_{m-1}) \overset{\text{def}}{=} \phi(E(t_{m-1})) = \text{true}$. Let $\langle \hat{h}_m, t_m \rangle = \langle \hat{h}_{m-1}, t_{m-1} \rangle \bowtie \langle r_m, \mathrm{pred}(r_m) \rangle$. By Lemma 3, we have $E(t_m) = E(\hat{h}_m)$ which implies $\phi(t_m) \overset{\text{def}}{=} \phi(E(t_m)) = \text{true}$. This implies $\langle \hat{h}_m, t_m \rangle \in H_W(E)$ as desired. □

In the case of the strict intersection filter function, this Proposition states that only hypotheses that cover every seed are generated. On the other hand, in the case of the support filter function, only hypotheses that cover the required number of seeds are generated.

In future work we will experimentally evaluate the effect of alternative settings of the parameters exposed for hypothesis space restriction by this framework. In particular, we want to observe their effect on the accuracy of learned hypothesis found under spaces restricted by Generalized Inverse Entailment. For example, determining what effect different support thresholds have on accuracy is important. Determining how robust this approach is to sampling effects as compared to Inverse Entailment would test the conjecture that the effect caused by a bad choice of a single seed is in fact mitigated by this proposed framework. Characterizing the types of datasets that benefit from this approach would be enlightening.

6 Conclusion

We presented a framework for ILP that exploits caching and avoids redundant computation. This framework is built upon a data structure and hypothesis extension operation that makes opportunities for caching explicit. We presented this structure and defined the extension operation in terms of relational database operations, suggesting a way to incorporate ILP in a relational database environment.

We also discussed how current methods that seek to improve efficiency and alternative search definition can directly benefit from the framework presented here. In addition, new variants of search restriction and strategy are direct results of this framework. We discussed one such variant, which generalizes Inverse Entailment to multiple seeds, and presented theoretical results that offer a foundation for this generalization.

Finally, this framework enables us to learn theories in languages other than sets of Horn clauses, including theories that make probabilistic statements, statements about aggregates, and that contain negation.

Each of the directions mentioned above holds the potential for significant improvement in some aspect of ILP, and we believe that the work in this paper is a first step that opens many promising avenues for future research.

References

1. Dzeroski, S., Lavrac, N., eds.: Relational Data Mining. Springer-Verlag New York, Inc. (2001)
2. Blockeel, H., Sebag, M.: Scalability and efficiency in multi-relational data mining. SIGKDD Explor. Newsl. **5** (2003) 17–30
3. Struyf, J., Blockeel, H.: Query optimization in inductive logic programming by reordering literals. In: ILP. (2003) 329–346
4. Bockhorst, J., Ong, I.M.: FOIL-D: Efficiently scaling FOIL for multi-relational data mining of large datasets. In: ILP. (2004) 63–79
5. Blockeel, H., Dehaspe, L., Demoen, B., Janssens, G., Ramon, J., Vandecasteele, H.: Improving the efficiency of inductive logic programming through the use of query packs. J. Artif. Intell. Res. (JAIR) **16** (2002) 135–166

6. Costa, V.S., Srinivasan, A., Camacho, R., Blockeel, H., Demoen, B., Janssens, G., Struyf, J., Vandecasteele, H., Laer, W.V.: Query transformations for improving the efficiency of ILP systems. Journal of Machine Learning Research **4** (2003) 465–491
7. Cussens, J.: Using prior probabilities and density estimation for relational classification. In: ILP. (1998) 106–115
8. Zelezný, F., Srinivasan, A., Page, D.: A Monte Carlo study of randomised restarted search in ILP. In: ILP. (2004) 341–358
9. DiMaio, F., Shavlik, J.W.: Learning an approximation to inductive logic programming clause evaluation. In: ILP. (2004) 80–97
10. Vens, C., Assche, A.V., Blockeel, H., Dzeroski, S.: First order random forests with complex aggregates. In: ILP. (2004) 323–340
11. Costa, V.S., Page, D., Qazi, M., Cussens, J.: CLP(BN): Constraint logic programming for probabilistic knowledge. In: International Conference on Uncertainty in Artificial Intelligence. (2003)
12. Srivasanan, A.: The Aleph manual. Source code available at http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html (2004)
13. Muggleton, S.: Inverse entailment and Progol. New Generation Comput. **13** (1998) 245–286
14. Ramakrishnan, R., Gehrke, J.: Database Management Systems. Third edn. McGraw-Hill (2003)
15. Ferilli, S., Mauro, N.D., Basile, T.M.A., Esposito, F.: Theta-subsumption and resolution: A new algorithm. In Zhong, N., Ras, Z.W., Tsumoto, S., Suzuki, E., eds.: ISMIS. Volume 2871 of Lecture Notes in Computer Science., Springer (2003) 384–391
16. Mauro, N.D., Basile, T.M.A., Ferilli, S., Esposito, F., Fanizzi, N.: An exhaustive matching procedure for the improvement of learning efficiency. [24] 112–129
17. Maloberti, J., Sebag, M.: Fast theta-subsumption with constraint satisfaction algorithms. Machine Learning **55** (2004) 137–174
18. Arias, M., Khardon, R.: Bottom-up ilp using large refinement steps. In Camacho, R., King, R.D., Srinivasan, A., eds.: ILP. Volume 3194 of Lecture Notes in Computer Science., Springer (2004) 26–43
19. Fonseca, N., Rocha, R., Camacho, R., Silva, F.M.A.: Efficient data structures for inductive logic programming. [24] 130–145
20. Yin, X., Han, J., Yang, J., Yu, P.S.: Crossmine: Efficient classification across multiple database relations. In: ICDE, IEEE Computer Society (2004) 399–411
21. Ramakrishnan, R., Srivastava, S., Sudarshan, S.: Efficient bottom-up evaluation of logic programs. In Dewilde, P., Vandewalle, J., eds.: Computer Systems and Software Engineering: State-Of-The-Art. Kluwer Academic Publishers (1992)
22. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996) 307–328
23. Dehaspe, L., Toivonen, H.: Discovery of frequent datalog patterns. Data Min. Knowl. Discov. **3** (1999) 7–36
24. Horváth, T., ed.: Inductive Logic Programming: 13th International Conference, ILP 2003, Szeged, Hungary, September 29-October 1, 2003, Proceedings. In Horváth, T., ed.: ILP. Volume 2835 of Lecture Notes in Computer Science., Springer (2003)