Chapter 4

RHB: ILP System that Learns Logic Programs with Sorts from Positive and Negative Examples

4.1 Problems in Previous ILP

We will discuss the problems experienced when top-down and bottom-up learners handle a large-scale sort hierarchy. Some previous learners were able to use types or type declarations for curtailing the search space. Their learning results, however, did not have sort information linked to these declarations.

4.1.1 Problems in Top-Down Learners

Top-down learners, such as FOIL, generate a number of tuples during clause construction when new variables are introduced. The more new symbols appear in the *is_a* relations, the slower FOIL becomes. As experimental results will show, FOIL is inherently slow when the background knowledge includes about 3000 *is_a* relations. Moreover, FOIL fails to learn rules with *is_a* relations, because it can not estimate the correct code length of clauses which is used as *the stopping criteria.*

A long chain of *is_a* relations also causes a problem with respect to the *variable depth*. If a sort hierarchy is 12 levels deep, the maximum variable depth must be at least 12. This extension greatly increases the search space for learning relations with sorts over those without sorts.

4.1.2 Problems in Bottom-Up Learners

Bottom-up learners, such as Golem, share the same problem with the variable depth as FOIL. When the variable depth is significant (*e.g.*, 12), Golem produces clauses that include a lot of literals by using Muggleton's *relative least generalization (lgg)*. Since the clauses include irrelevant literals, Golem heuristically reduces the clauses. However, this process often drops *is_a* relations and takes much time [1] when the background knowledge is large. Progol has the same problem as Golem, although it can handle type declarations described in a clausal form. As a result, Golem and Progol virtually fail to effectively produce clauses that include the necessary *is_a* relations.

If *is_a* relations represent direct links in a sort hierarchy as well as indirect general-specific relations of two sorts, they are no longer *determinate*. Golem uses determinate literals to reduce the search space and are literals whose terms are determinate terms. For example, $is\_a(X, male)$ is a *determinate literal* if the background knowledge includes only $\{is\_a(Peter, male),\ is\_a(male, human)\}$; however, once $is\_a(Jack, male)$ is added, *is_a* becomes a *non-determinate literal* because the varialbe X of a literal $is\_a(X, male)$ has possible bindings to both $Peter$ and $Jack$.

4.2 System Overview

The Relational Learner with Hierarchical Background Knowledge (RHB) [54] generates order-sorted Prolog programs that discriminate between positive and negative examples on the basis of background knowledge that includes a large-scale sort hierarchy.

[1] Golem requires the input-output mode of *is_a* in order to reduce the clause reduction time. However, the mode declaration of *is_a* is a big hint because the two arguments of *is_a* can be both input and output.

RHB provides sorts with special operations. RHB is implemented with a LIFE programming language [2] , and it can efficiently learn relations with the sort hierarchy through its sort handling mechanisms.

4.2.1 Framework

The framework of hierarchically sorted ILP based on τ-terms is defined as follows. In this chapter, we call literals and clauses based on τ-terms just literals and clauses.

Let a signature be $\Sigma_{OSF} = \langle \mathcal{P}, \mathcal{S}, \preceq, \sqcap, \sqcup, \mathcal{L} \rangle$, where $\mathcal{P}$ is a finite set of predicates, $\mathcal{S}$ is a finite set of sort symbols, and $\mathcal{L}$ is a finite set of feature symbols and $\mathcal{V}$ be a finite set of variables.

Example language L_E, background knowledge language L_B, and hypothesis language L_H are defined as follows.

- L_E: the set of all ground atomic formulae whose predicate symbols is an observation predicate symbol p.
- L_B: the set of all ground unit clauses without the predicate symbol p.
- L_H: the set of all definite clauses whose heads are atomic formulae with p and whose bodies consist of literals with predicates symbols in background knowledge language L_B.

Let a finite set of positive examples be $E^+ \subseteq L_E$, a finite set of negative examples be $E^- \subseteq L_E$, and a finite set of background knowledge be $B \subseteq L_B$, where the following conditions are satisfied:

$$\forall e \in E^+ \; B \not\models_{OSF} e$$

$$\forall e \in E^- \; B \models_{OSF} e$$

Hierarchically sorted ILP based on τ-terms is defined as φ that satisfies the following conditions with respect to hypotheses H.

$$H \subseteq L_H \;\; s.t. \;\; H = \varphi(L_E, L_B, L_H, B, E^+, E^-).$$

[2] wild_life, an implementation of LIFE, is available at `http://www.isg.sfu.ca/life`.

$$B \cup H \text{ is consistent.}$$
$$\forall e \in E^+ \; B \cup H \vdash_{OSF} e.$$
$$\forall e \in E^- \; B \cup H \not\vdash_{OSF} e.$$

Because we apply RHB to experiments on the learning from real-world data with some noise, the last two conditions should be relaxed to "for most of $e \in E^+$ and for most of $e \in E^-$". We employ the model complexity measure [13] as a heuristics to estimate the appropriateness of H.

4.3 Model Complexity Measure

The *proof complexity measure* [37] is the total complexity of choice points in an implementation of *SLDNF resolution*[32]. Conklin and Witten [13] illustrated that the *model complexity measure* can estimate a more appropriate code length of the hypothesis clauses in two example problems than the proof complexity measure. Since both complexity measures require the computation of the code length of a set of clauses, we will explain a complexity measure of clauses with sorts based on the model complexity measure. Example 1, then, demonstrates the serious problem of treating *is_a* relations as ordinary background knowledge through the code length measure.

According to Conklin and Witten [13], *complexity-based induction* with model complexity finds the best set logic program T for a set of examples E so that the description length $L_{\mathcal{MC}}(T|E)$ is minimized.

Definition 68 (code length of an atom) *Given a signature* $\Sigma = (\mathcal{F}, \mathcal{P}, \mathcal{C})$, *the code length of an atom with the predicate symbol* p *is*

$$\log_2(|\mathcal{P}|) + n \log_2(|\mathcal{V}| + |\mathcal{C}|).$$

Definition 69 (code length of a theroy) *The code length* $L(T)$ *is the sum of the following bits:*

- $\log_2(v+1)$ *bits where* v *is the number of variables in the program.*
- *1 bit per program, 2 bits per rule in the program, and 2 bits per literal in the body of each rule.*

- *the bits for all atoms in the program.*

Definition 70 (description length) *$L_{\mathcal{MC}}(T|E)$ is defined as:*

$$L_{\mathcal{MC}}(T|E) \stackrel{\text{def}}{=} L(T) + L_{\mathcal{MC}}(E|T).$$

$L_{\mathcal{MC}}(E|T)$ denotes the length of examples E with respect to a logic program T, which is measured by the model complexity.

Definition 71 (code length of examples) *$L_{\mathcal{MC}}(E|T)$ is defined as*

$$L_{\mathcal{MC}}(E|T) \stackrel{\text{def}}{=} \log_2 \binom{|Q(T)|}{|E|},$$

where $Q(T)$ is an empiracal content.

Note that, if $|Q(T)| = |E|$ then $L_{\mathcal{MC}}(E|T)$=0.

4.4 Code Length of τ-terms

In order to compute the code length of clauses in $\mathcal{L}_\tau$, we extend the definition of the code length of an atom as follows.

Definition 72 (code length of an atom) *Given a signature $\Sigma_{OSF} = \langle \mathcal{P}, \mathcal{S}, \preceq, \sqcap, \sqcup, \mathcal{L} \rangle$, the code length of an atom with the predicate symbol p is*

$$\log_2(|\mathcal{P}|) + n \log_2(|\mathcal{V}| \times |\mathcal{S}|).$$

Since a τ-term has sorts attached to variables, we should state the sort associated with each variable. The following example demonstrates the computation of the code length for the hypothesis clause of "mortal".

Example 5

The code lengths of three simple definitions of *mortal* are compared. Suppose that T_i are the results of learning from the following examples:

Table 4.1: Code Length of Example Clauses

T_i	$L(T_i)$	$L_{mc}(E\|T_i)$
(T_1) $mortal(X : human)$	$\log_2(2)$+1+ 2 + 0 +1×(1+$\log_2$(1×3000))=16.6	0
(T_2) $mortal(X)$:- $X < Y$, $Y < Z$, $Z < human$.	$\log_2(4)$+1+ 2 +2×3 +3×(1 + 2×$\log_2$(3+3000+7))=83.3	0
(T_3) $mortal(p1)$. $mortal(p2)$. $mortal(p3)$. $mortal(p4)$. $mortal(p5)$.	0 + 1 + 2×5+ 0 + 5×(1 + $\log_2$(0 + 3000+7)) = 73.8	0

- positive: { *mortal(p1),mortal(p2),mortal(p3),mortal(p4), mortal(p5)* },
- negative: { *mortal(t1), mortal(t2)* }.
- background knowledge: { *p1*$\preceq$*male, p2*$\preceq$*male, p3*$\preceq$*male, p4*$\preceq$*female, p5*$\preceq$*female, male*$\preceq$*male-female, female*$\preceq$*male-female,*

  male-female$\preceq$*human, t1*$\preceq$*thing, t2*$\preceq$*thing, ...*

  human$\preceq$*anything, thing*$\preceq$*anything,*

  (In total, 3000 $\preceq$ *relations are written here.)* }

The background knowledge consists of 3000 *is_a* relations denoted by $\preceq$.

Let us compare code lengths of traditional logic programs and a traditional logic program based on τ-terms. Table 4.1 shows that the code lengths of logic programs T_1, T_2 and T_3, where T_1 is a logic program based on τ-terms, T_2 is a traditional logic program, and T_3 is examples.

The code lengths $L(T_i)$ of T_1, T_2, and T_3 are 16.6 bits, 83.3 bits and 73.8 bits, respectively. $L_{\mathcal{MC}}(E|T_i)$ is zero because the model of

T_i exactly matches that of the original examples. T_2 is rejected as a hypothesis clause because the bit length of T_2 exceeds that of the examples (*i.e.*, T_3). This strongly suggests that hypothesis languages should involve sorts and that *is_a* relations should be treated separately from other background knowledge.

As explained in this section, the code length measure should be extended to deal with a sort hierarchy, and *is_a* relations should be treated as special background knowledge in computing the appropriate clause complexity.

4.5 Algorithm of RHB

RHB learns hypotheses in a combined bottom-up and top-down manner, following the result presented in [65]. Namely, the head is made in a bottom-up manner and then the body is constructed in a top-down manner.

The outer loop of RHB finds covers of the positive examples E^+ in a greedy manner. It constructs hypothesis clauses one by one by calling *inner_loop*(E^+, B) (Algorithm 1), which returns a hypothesis clause. Covered examples are removed from E^+ in each cycle.

In the following sections, we will now see how sorts are utilized in each component of the RHB algorithm.

4.5.1 Least General Generalization of τ-terms

A least general generalization of two τ-terms is defined as:

- $lgg(X{:}s, X{:}s) = X{:}s$.
- $lgg(X{:}s, Y{:}t) = Z{:}s \sqcup t$.

From the definition of $\sqcup$, it is clear that the result of lgg is a least general generalization of two terms with respect to ordering of clauses $\leq$. Other definitions of lggs of literal and clauses are the same as Plotkin's lggs.

Algorithm 1 *inner_loop*(E^+, B)

1 Given positive E^+ *and* E^-*, background knowledge* B*.*

2 Determine the sorts of variables in a head by computing sorted lggs of k *pairs of elements in* E^+ *for predefined constant* k*, and select the most general head as Head.*

3 If the stopping condition is satisfied, return Head.

4 Let Body be empty.

5 Create a set of all possible literals L*.*

6 Select the literal l *with the highest information gain as evaluated by the model complexity measure.*

7 Add l *to Body.*

8 Dynamically restrict sorts in Head :– Body.

9 If the stopping condition is satisfied, return the clause Head :– Body.

10 Goto 5.

4.5.2 Dynamic Sort Restriction

The special feature of RHB is its *example-guided* sort restriction of a clause during clause construction. After adding a literal, for each root variable X_i of τ-terms $X_i{:}s_{X_i}$ in the current clause, RHB computes $\tau^+_{X_i}$ that is the *lub* of all sorts that are matched to s_{X_i} when unifying each covered positive examples with the current head and prove the body. In the same way, it computes $\tau^-_{X_i}$ which is the *lub* of all sorts matched to s_{X_i} when unifying each covered negative examples with the current head and prove the body. Then, the following dynamic sort restriction is applied for all i and obtain a new *Body*.

Definition 73 (Dynamic Sort Restriction)

1. *If* $\tau^+_{X_i} \prec \tau^-_{X_i}$ *then replace* $X_i{:}s_{X_i}$ *in* $Head :- Body$ *by* $X_i{:}\tau$, *where* τ *is the most general sort such that* $\tau^+_{X_i} \preceq \tau \preceq \tau^-_{X_i}$.
2. *If* $\tau^-_{X_i} \preceq \tau^+_{X_i}$ *then replace* $X_i{:}s_{X_i}$ *in* $Head :- Body$ *by* $X_i{:}\tau^+_{X_i}$.
3. *If the order of* $\tau^+_{X_i}$ *and* $\tau^-_{X_i}$ *is undefined then replace* $X_i{:}s_{X_i}$ *in* $Head :- Body$ *by* $X_i{:}\ \tau^+_{X_i}$.

According to this sort restriction, newly introduced variables in the body can be bound to sorts at certain levels of the sort hierarchy. If RHB had no such sort restriction, newly introduced variables would always be bound to $\top$, and RHB could produce overgeneral clauses. Moreover, the result of the sort restriction operation by the unification dynamically affects the sorts of all variables related to the unified variable. LIFE's sort unification mechanism directly performs this operation.

4.6 Experiments and Results

Two kinds of experiments were conducted with 3000 *is_a* relations representing direct links in a sort hierarchy to confirm that RHB can efficiently handle a sort hierarchy while still achieving a high accuracy. The experiment was performed on a SparcStation 20 with a main memory of 96 Mbyes.

4.6.1 Learning Time and Sort Hierarchy Size

To estimate the effect of the sort hierarchy size on the learning speed, the sample relation "speak" was learned several times by FOIL, Golem, Progol and RHB, for different numbers of *is_a* relations.

Figure 4.1 shows that FOIL becomes inherently slow when learning the relation with about 3000 *is_a* relations. On the other hand, the learning speed of RHB, Golem and Progol were not affected by the number of *is_a* relations. [3] However, Golem needed an input-output

[3] The current version of RHB runs on a LIFE interpreter while the other learners are written in C.

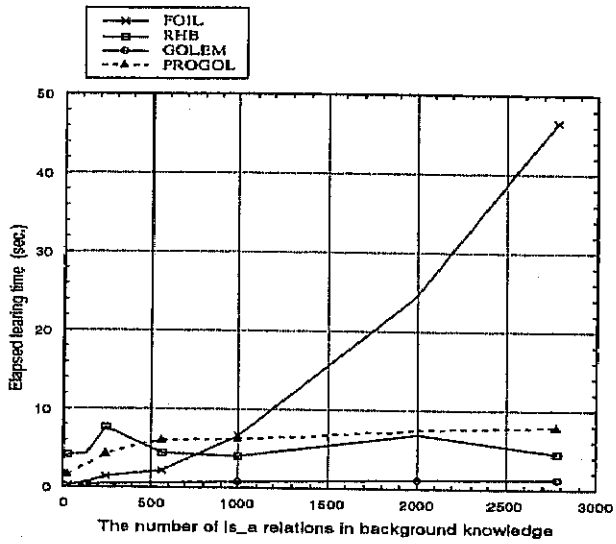

Figure 4.1: Learning Time vs Sort Hierarchy Size

mode declaration for *is_a* relations. Without these declarations, for example, Golem took 209.8 sec with the 3000 *is_a* relations. Golem's clause reduction phase spent most of the time searching for effective *is_a* relations.

4.6.2 Comparison of Accuracy

To examine the accuracy, five sets of 20, 50, 100 and 200 examples are prepared. The examples were randomly generated where the positive examples satisfied, but the negative examples did not satisfy, the following *answer clause*:

$$speak(A : person, B : language) :-$$

$$grew(A, C : country), official\_lang(B, C).$$

Learning experiments were conducted on the 20, 50, 100 and 200 training examples and on 200 test examples different from the training examples with 3000 total *is_a* relations. The learners were RHB,

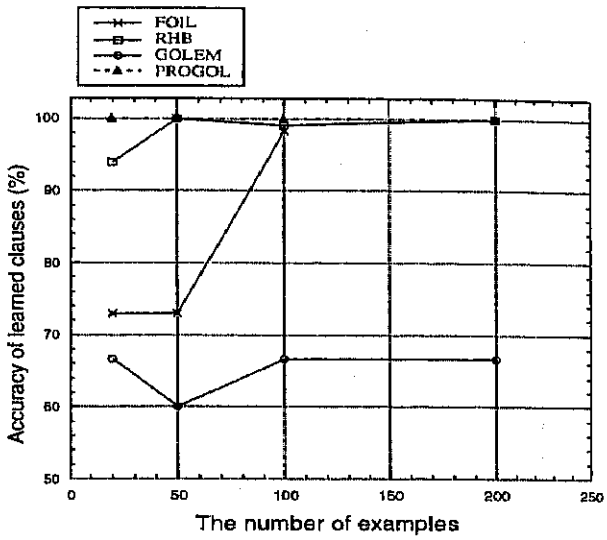

Figure 4.2: Accuracy vs Example Size

FOIL[4] , Golem and Progol. Figure 4.2 shows the accuracy of RHB, FOIL, Golem and Progol. RHB and Progol exhibited a high accuracy. FOIL and Golem produced overgeneral clauses while RHB and Progol produced clauses almost identical to the answer clause. Figure 4.3 shows the time taken in the learning in seconds. Compared to Golem, RHB seemed to learn slowly, but without input-output mode declarations, Golem spent 4281 sec. to learn from 100 examples. Progol was two times slower than RHB. These results indicate that introducing sorts into ILP systems enables not only efficient learning, but also learning with a high accuracy.

4.7 Related Work

The lgg of ψ-terms has already been illustrated in [1]. The generalization for *Sorted First Order Predicate Calculus (SFOPC)* [15] is presented in [16]. The generalization algorithm for SFOPC can achieve

[4] FOIL failed to learn from 200 examples because of insufficient memory.

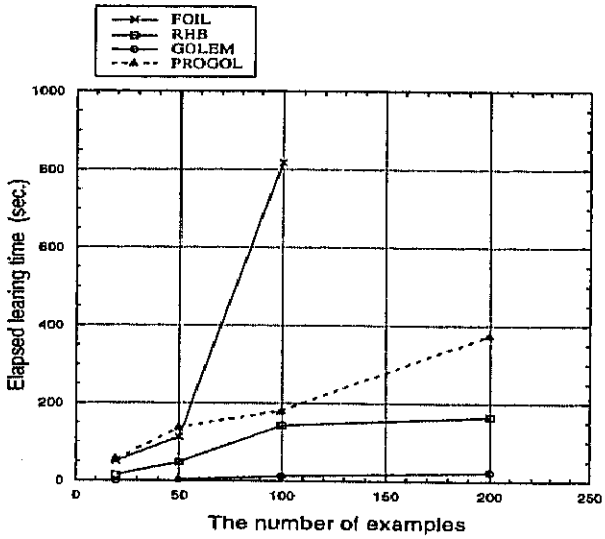

Figure 4.3: Learning Time vs Example Size

the τ-term lgg by converting formulae and sort hierarchy to the *A-expression* and *S-expression* of the SFOPC, respectively. A feature of our *operational* definition of the ψ-term lgg is that the algorithm is given as a partial extension of the definition of Plotkin's lgg.

4.8 Summary

This chapter presented the Relational Learner with Hierarchical Background Knowledge (RHB). It generates sorted Prolog programs that discriminate between positive and negative examples on the basis of background knowledge that includes a large-scale sort hierarchy as hierarchical background knowledge. Previous learners, such as FOIL, Golem and Progol, have serious problems in handling relations with a large-scale sort hierarchy. RHB provides *is_a* relations with special operations when computing *lggs* and the code length of clauses. RHB was implemented in LIFE, which can efficiently learn relations with a sort hierarchy through its sort handling mechanisms. Experimental results

showed that RHB can efficiently handle about 3000 *is_a* relations while still achieving a high accuracy.

Sample problem "speak"

```
% positive examples

speak(jack, english).
speak(betty, japanese).
speak(taro, japanese).
speak(jiro, japanese).
speak(goro, japanese).
speak(cow1,moo).
speak(cow2,moo).

% negative examples

speak(cow1,japanese).
speak(cow1,english).
speak(cow2,japanese).
speak(cow2,english).
speak(taro,moo).
speak(jiro,moo).
speak(jack, japanese).
speak(betty, english).
speak(taro, english).
speak(jiro, english).
speak(goro,  english).

% background knowledge

born(jack, america).
born(betty, america).
born(taro, japan).
born(jiro, japan).
born(goro, america).
born(moo, japan).
born(mou, america).

grew(jack, america).
grew(betty, japan).
grew(taro, japan).
```

```
grew(jiro, japan).
grew(goro, japan).
grew(cow1, japan).
grew(cow2, america).

official_lang(america, english).
official_lang(japan, japanese).

jack < old_aged.
taro < brother.
jiro < husband.
goro < male.
betty < female.
america < country.
japan < country.
english < language.
japanese < language.
moo  < abstract.
cow1 < animal.
cow2 < animal.
...
% other is_a relations.
```