Chapter 5

RHB$^+$: ILP System that Learns Logic Programs with Sorts from Positive Examples

This chapter describes the novel relational learner RHB$^+$ [56] which generates sorted Prolog programs from just positive examples on the basis of background knowledge which might include a large-scale sort hierarchy.

5.1 Overview

The hypothesis language of RHB$^+$ is the Horn clause based on τ-terms, same as RHB. The hierarchically sorted ILP system has the following features that match the needs for learning IE rules.

- A hierarchically sorted ILP system can efficiently and effectively handle sort (or semantic category) information in training data. This feature is advantageous in controlling the generality and accuracy of learned IE rules.
- It can directly use semantic representation of the text as background knowledge.
- It can learn from only positive examples.

- Predicates are allowed to have features (or keywords) for readability and expressibility.

5.1.1 Framework

The framework of hierarchically sorted ILP based on τ-terms that learns from only positive examples 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 $E^+ \subseteq L_E$ and a finite set of background knowledge be $B \subseteq L_B$, where the following conditions are satisfied:

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

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

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

$$B \cup H \ is \ consistent.$$

$$For \ most \ of \ e \in E^+ \quad B \cup H \vdash_{OSF} e.$$

$$For \ most \ of \ e \in L_E - E^+ \quad B \cup H \not\vdash_{OSF} e.$$

Algorithm 2 *outer_loop*(E^+, B)

```
1. Preserve original positive examples $E^+$ to $E_0^+$.
2. $Hypo \leftarrow \{\}$.
3. $H \leftarrow inner\_loop(E^+, E_0^+, B)$.
4. If $H$ is empty, $Hypo \leftarrow Hypo \cup E^+$ and return $Hypo$.
5. Otherwise, $Hypo \leftarrow Hypo \cup \{H\}$.
6. Removed positives covered by $H$ from $E^+$.
7. If $E^+$ is empty, return $Hypo$.
8. Goto Step 3.
```

Because our goal is to apply RHB$^+$ to the learning from real-world data with some noise, the second last condition is relaxed from "$\forall e \in E^+$" to "for most of $e \in E^+$". The last condition states that H is more preferable if it covers a smaller amount of examples in L_E other than E^+. We employ in this chapter the weighted informativity as a heuristics to estimate the appropriateness of H.

5.2 Algorithms of RHB$^+$

RHB$^+$ employs a combination of bottom-up and top-down approaches, following the result described in [65]. That is, first make the head in a bottom-up manner then construct the body in a top-down manner.

The outer loop of RHB$^+$ finds covers of the given positive examples E^+ in a greedy manner (Algorithm 2). It constructs clauses one by one by calling *inner_loop*(E^+, E_0^+, B) (Algorithm 3), where E_0^+ is original positive examples and B is background knowledge. Covered examples are removed from E^+ in each cycle; E_0^+ remains unchanged. The *inner_loop* calls make_body (Algorithm 4) which constructs the body of a hypothesis clause.

Algorithm 3 *inner_loop*(E^+, E_0^+, B)

```
1. Randomly select N pairs of positives and compute lggs of the pairs. Select one lgg that covers the most positives as Head. If there are some candidates, select one lgg at random.
2. If StoppingCondition($E_0^+$, B, Head) is satisfied, return Head.
3. Body ← {}, NewBody←{}.
4. Call make_body($E^+$, $E_0^+$, B, Head, Body, NewBody). If the result is fail, return {}.
5. Otherwise, return Head :- NewBody.
```

We will now see how sorts are utilized in each component of the RHB$^+$ algorithm in the following sections.

5.2.1 Dynamic Sort Restriction by Positive Examples

The special feature of RHB$^+$ is the *dynamic sort restriction by positive examples* during clause construction. *restrict*$(E^+, B, (Head :- Body))$ in Algorithm 5 does this part, where E^+ represents positives, B is background knowledge, and (*Head*:- *Body*) is a hypothetical clause. The restriction uses positive examples currently covered in order to determine appropriate sorts. Informally, for each variable X_i appearing in the clause, RHB$^+$ computes the *lub* of all sorts bound to X_i when covered positive examples are unified with the current head in turn. Formally, the dynamic sort restriction by positive examples is defined as follows.

Sort restriction replaces sorts of newly introduced variables in the body. Without a sort restriction, newly introduced variables would always have no sorts, and RHB$^+$ might produce over-general clauses. Note that the result of the sort restriction operation by unification dynamically affects the sorts of all variables related to the unified variable.

Algorithm 4 *make_body*$(E^+, E_0^+, B, Head, Body, NewBody)$

```
1. Create a set of all possible literals $L$ using variables in $Head$ and
   $Body$, predicates in $B$ and new variables.
2. Remove literals already occurred in $Body$.
3. If $L$ is empty or the number of literals in $Body$ exceeds the limit
   $d$, $NewBody \leftarrow \{\}$ and return fail.
4. Set top $K$ literals $l_k \in L$ to list $BEAM$, by evaluating weighted
   informativity $\underline{PWI(E^+, B \cup \{Head\ :-\ Body, l_k\})}$.
5. If $BEAM$ is empty, goto Step 6; otherwise, pick out the first
   element of $BEAM$ and set it to $l$. Do the following steps for
   $Body_l \leftarrow Body, l$.
   5-1. Restrict the sorts of variables in the current hypothesis by
        $\underline{restrict(E^+, B, (Head:-Body_l))}$.
   5-2. If $\underline{StoppingCondition(E_0^+, B, (Head:-Body_l))}$ is satisfied,
        $NewBody \leftarrow Body_l$ and return true.
   5-3. Call make_body$(E^+, E_0^+, B, Head, Body_l, NewBody)$. If the
        result is true, return true; otherwise, goto Step 5.
6. $NewBody \leftarrow \{\}$ and return fail.
```

This operation is directly implemented by using the sort unification mechanism of LIFE.

It would rather not aggressively add sort to narrow the current cover but it is interesting that the dynamic sort restriction significantly contributes to narrowing the current cover and helping the learner to find good hypotheses.

Example 6

When we have positive examples and background knowledge as given in Example 1 and additional data about cat Socks.

Algorithm 5 (*Dynamic sort restriction by positive examples*)

1. *Given a hypothesis clause Hypo = (Head :−Body) and positives* E^+.
2. *Collect all the terms* $X_k{:}s_k$ *in Hypo and put* X_k *into a list VarSet.*
3. *Let* $\hat{E}^+$ *be examples in* E^+ *covered by Hypo.*
4. *For all elements* p_i *of* $\hat{E}^+$, *unify Head and* p_i, *then prove Body. Make a list* $SortSet_i$ *of bound sorts in the proven Head and Body so that the position of each sort in* $SortSet_i$ *correctly corresponds to the position of the original variables in VarSet.*
5. *For all* X_k *in VarSet, compute lub* τ_k *of all bound sorts of* X_k *in* $SortSet_i$.
6. *For all k, replace* $X_k{:}s_k$ *by* $X_k{:}\tau_k$.

- *Additional positive example :*
  { *speak(Socks,cat-lang)* }

- *Additional background knowledge :*
  {*grew_in(Socks,Japan)*, *Socks* < *cat*}.

Suppose that *speak(agent,anything)* is the current head. Adding *official_lang*/2 , one of the candidates for additional literal, restricts the sorts in the head as follows. The current clause before the sort restriction is:

speak(agent, Y:anything) :– *official_lang*(X, Y).

The second argument of *official_lang* matches an official language. This cause that covered positives are only examples relating to humans because the positive related to *Socks* is no longer covered. Therefore, the sort *agent* is restricted to humans and we obtain:

speak(human, Y:language) :– *official_lang*(X,Y).

After that, the data unrelated to humans will not affect the clause construction. This illustrates how the typing contributes to restricting sorts in forming sorted clauses.

5.2.2 Use of Sorts in Computing Informativity

Sort information is also used to compute positive weighted informativity $PWI(E^+, B \cup \{Head :\!-Body\})$. Let $T = B \cup \{Head :\!-Body\}$.

$$PWI(E^+,T) = -\frac{1}{|\hat{E^+}|} \times \log_2 \frac{|\hat{E^+}|+1}{|Q(T)|+2},$$

where $|\hat{E^+}|$ denotes the number of positive examples covered by T. In effect, this is weighted informativity employing the Laplace estimate [11].

Sort information is very useful for computing $Q(T)$. Let Hs be a set of instances of *Head* generated by proving *Body* using backtracking. We introduce the notation $|\tau|$ which expresses the number of constants that sort τ represents. When τ is a constant, $|\tau|$ is defined as 1. Intuitively, $|\tau|$ is the number of constants under τ in the sort hierarchy.

$$|Q(T)| = \sum_{h \in Hs} \prod_{\tau \in Sorts(h)} |\tau|,$$

where $Sorts(h)$ returns the set of sorts in h.

Example 7

Positive examples E^+ and background knowledge are as given in Example 5. T is the background knowledge and the following clause:

speak(*human*, Y: *language*) :− *official_lang*(X,Y).

In this case, the set of instances Hs is $\{speak(human, Japanese), speak(human, English)\}$. Caution is needed in that *human* is the sort representing two constants: *Jack* and *Jun*.

$$|Q(T)| = |human| \times |Japanese| + |human| \times |English|$$
$$= 2 \times 1 + 2 \times 1 = 4.$$

Sort information drastically reduces the time to calculate informativity because it cuts the effort of generating huge numbers of combinations of constants for computing the empirical content. In this case, $PWI(E^+,T)$ is calculated as follows:

$$PWI(E^+,T) = -\frac{1}{2} \times \log_2 \frac{2+1}{4+2} = 0.5.$$

5.2.3 Use of Sorts in the Stopping Condition

This section describes how the stopping condition works. When given only positive examples, usual informativity cannot be applied. When T is the current hypothesis and background knowledge, we use the *Model Covering Ratio (MCR)*:

$$MCR(T) = \frac{|\hat{E}_0^+|}{|Q(T)|}.$$

The stopping condition is "$MCR(T) \geq \alpha$" for a predefined constant α ($0 \leq \alpha \leq 1$). Note that $|\hat{E}_0^+|$ here denotes the original positive examples covered by T. This is because $|Q(T)|$ may include a lot of examples removed from E^+ in earlier steps . Sort information plays a key role in computing the stopping condition because it permits the efficient calculation of $|Q(T)|$ as described in the previous section.

5.3 Experiments and Results

In order to confirm that RHB$^+$ can efficiently handle a sort hierarchy, two kinds of experiments were conducted with one part of 3000 *is_a* relations. We selected more appropriate representation of *is_a* relations for each learner: Progol incorporated *is_a* literals, which represent direct links in a sort hierarchy, in background knowledge and FOIL used sort literals.

The first experiment determined the effect of sort hierarchy size. We tested FOIL, Progol [1] and RHB$^+$ on artificial data while changing hierarchy size.

The second experiment measured the performance of those three learners with real data extracted from newspaper articles. We used a SparcStation 20 with 96 Mbyes of memory for the experiments.

5.3.1 Learning Time and Sort Hierarchy Size

In order to estimate the effect of sort hierarchy size on learning speed, we randomly generated positive examples that satisfied the following *answer clause*:

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

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

Figure 5.1 shows that FOIL exponentially slows as the sort hierarchy size increases. On the other hand, the learning speed of RHB$^+$ and Progol were not affected so much by the number of *is_a* relations. MCRs of the results from the artificial data should not be taken seriously.

[1] Progol4.2 with `set(posonly)` option.

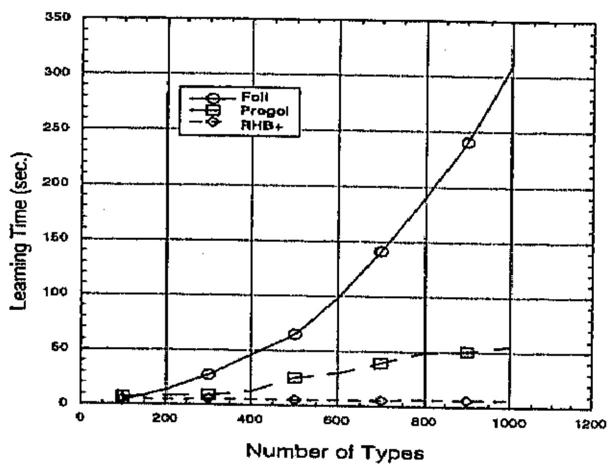

Figure 5.1: Learning Time vs Sort Hierarchy Size

5.4 Related Work

Some previous learners, such as FOIL [46], Golem [35] and Progol [40], use types or type declarations for curtailing the search space. Their learning results, however, do not have type information linked to those declarations. Simply including sorts and *is_a* relations in background knowledge is not a solution to obtaining sorted clauses. The reason is that the possibility of a long *is_a* or sort chain creates excessive overhead; the learner must search for all *is_a* literals or sort literals. For example, when the sort hierarchy is twelve levels deep, a chain of up to twelve *is_a* literals should be checked. When *is_a* represents direct links in a sort hierarchy, one possible chain might be:

$$is\_a(X,Y), is\_a(Y,Z), ..., is\_a(W,V), is\_agent(V), ...$$

When *is_a* includes indirect links in a sort hierarchy, atoms to be checked are:

$$is\_a(X, male), is\_a(X, human), ..., is\_a(X, agent), ...$$

When sort literals represent sorts in a sort hierarchy, atoms to be checked are:

$male(X), human(X), ..., agent(X), ...$

In those cases, top-down learners spend too much time trying to construct those chains while bottom-up learners try to remove some of *is_a* or sort atoms and to find good hypotheses.

Special treatment to sorts was presented in [64]. It requires both positive and negative examples to efficiently decide *one_isa* atoms.

According to an input-output declaration, FOIDL [33] generates implicit negatives by *output queries* for input arguments of positive examples in a normal ILP setting. RHB$^+$ utilizes sort information to compute the number of covered examples including implicit negatives.

5.5 Summary

RHB$^+$, which learns sorted Prolog programs, was presented. Its performance is not affected by the number of sorts or sort hierarchies size for the sake of direct manipulation of sorts and utilization of sort information in computing informativity heuristics and stopping conditions. It also achieved appropriate generalization levels of hypotheses. At this point, a full LIFE compiler is not available but the current interpretive version of RHB$^+$ showed good performance. The execution speed will markedly improved when a LIFE compiler becomes available.