Chapter 7

Sample Trace of the Three Learners

7.1 RHB

This section illustrates a sample trace of RHB. Suppose that we have the following examples and background knowledge.
Positive examples:

```
happy(mary).
happy(john).
```

Negative example:

```
happy(jack).
```

Background knowledge:

```
id(p1,mary).
id(p2,john).
id(p3,jack).

got(p1,money).
got(p2,job).
got(p3,debt).
```

```
mary < person.
john < person.
jack < person.

p1 < code.
p2 < code.
p3 < code.

money < valuable.
job < valuable.
debt < non_valuable.

valuable < thing.
non_valuable < thing.

thing < everything.
person < everything.
code < everything.
```

First, RHB computes the lggs of positive examples. In this example, $happy(person)$ is the only lgg of pairs of positives. The head is set to

$$happy(person).$$

RHB then constructs the body. Predicates $id/2$ and $got/2$ are candidates for the literal to be added. RHB tries all possible literals generated by combining the predicates, variables, and sorts of the head and new variables and then adds the best literal in terms of the model complexity to give

$$happy(P:person) :- id(X, P).$$

After the sort restriction, X is restricted to the sort $code$:

$$happy(P:person) :- id(X:code, P).$$

Since the current hypothesis covers a negative example, the construction of the clause continues. By finding the best literal to be added again, RHB builds

$$happy(P:person) :- id(X:code, P), got(X, Y).$$

After restricting the sorts, it obtains

$$happy(P:person) :-$$

$$id(X:code, P), got(X, Y:valuable).$$

Since this clause covers no negative example, RHB finishes the learning process and outputs the result.

7.2 RHB$^+$

RHB$^+$ learns logic programs with sorts in the same way as RHB but uses only positive examples. Using the same example in the previous section, let us consider the sample trace of RHB$^+$. First, it computes the lggs of positive examples. In this example, $happy(person)$ is the only lgg of pairs of positives. The head is set to

$$happy(person).$$

RHB$^+$ then constructs the body. Predicates $id/2$ and $got/2$ are candidates for the literal to be added. RHB$^+$ tries all possible literals generated by combining the predicates, variables, and sorts of the head and new variables and then adds the best literal in terms of the PWI to give

$$happy(P:person) :- id(X, P).$$

After the sort restriction by only positives, X is restricted to the sort *code*:

$$happy(P:person) :- id(X:code, P).$$

Since the current hypothesis does not satisfy the stopping condition MCR, *i.e.*, $happy(jack)$ can be generated from the current hypothesis, the construction of the clause continues. By finding the best literal to be added again, RHB$^+$ builds

$$happy(P:person) :- id(X:code, P), got(X, Y).$$

After restricting the sorts, it obtains

$$happy(P:person) :-$$

$$id(X:code, P), got(X, Y:valuable).$$

Since this clause coverage satisfies the stopping condition, RHB$^+$ finishes the learning process and outputs the result.

7.3 ψ-RHB$^+$

ψ-RHB$^+$ takes a very different approach: it computes rlggs from positives and background knowledge.

From $happy(mary)$, it first selects $id(p1, mary)$ since the literal contains $mary$. Then, if the variable depth is two, ψ-RHB$^+$ selects $got(p1, money)$ from the background knowledge since the literal contains $p1$ which appears in $got(p1, money)$. In the same way, ψ-RHB$^+$ selects $id(p2, john)$ and $got(p2, job)$ for $happy(john)$. After linking the same sorts. It then computes an lgg of the following two clauses:

$$happy(X : mary) :- id(P : p1, X), got(P, money),$$

$$happy(Y : john) :- id(Q : p2, Y), got(Q, job).$$

As a result, the following clause is obtained:

$$happy(Z : person) :- id(R : code, Z), got(R, valuable).$$