UNIVERSIDADE DE SÃO PAULO

Learning Horn clauses using the ILP system

GOLEM

MARIA DO CARMO NICOLETTI

MARIA CAROLINA MONARD

Nº 4

N O T A S

Instituto de Ciências Matemáticas de São Carlos

Instituto de Ciências Matemáticas de São Carlos

ISSN - 0103-2577

Learning Horn clauses using the ILP system GOLEM

MARIA DO CARMO NICOLETTI

MARIA CAROLINA MONARD

Nº 4

N O T A S D O I C M S C

Série Computação

São Carlos

nov./ 1993

Learning Horn Clauses using the ILP System GOLEM

Maria do Carmo Nicoletti[1]
Universidade Federal de São Carlos/ILTC/IFQSC-USP
Departamento de Computação
Via Washington Luiz, km 235
Caixa Postal 676, 13565-905 - São Carlos, SP
e-mail:carmo@icmsc.usp.br

Maria Carolina Monard[2]
Universidade de São Paulo/ILTC
Instituto de Ciências Matemáticas de São Carlos
Departamento de Ciências de Computação e Estatística
Caixa Postal 668, 13560-970 - São Carlos, SP
e-mail: mcmonard@icmsc.usp.br

Abstract

Inductive Logic Programming — ILP — integrates the techniques already available and established for logic programming in a framework of learning, aiming to induce first-order logic programs from examples, using background knowledge.
The inductive logic programming system called *GOLEM* — due to Stephen Muggleton and Cao Feng — constructs logic programs from positive and negative examples using information given by the background knowledge. In *GOLEM* the examples and the background knowledge are represented as ground atomic facts. The purpose of this work is to present the main conceptual notions used by *GOLEM*, to show in details its ways of operation, to discuss the "quality" of the positive and negative examples required for inducing the appropriate expression of the concept, to evidence some of its limitations and show some of its results.

[1]Work partially supported by State Research Council FAPESP Proc. Nº 92/2151-8.
[2]Work partially supported by National and State Research Council — CNPq and FAPESP Proc. Nº 92/2151-8.

Contents

1 Introduction

Inductive Logic Programming — ILP — integrates the techniques already available and established for logic programming in a framework of learning, aiming to induce first-order logic programs from examples, using background knowledge.

The foundation of ILP — although not restricted to Horn clauses logic programming such as Prolog — were laid by Plotkin [Plotkin 71] through two major contributions

1. the introduction of a relationship of generality between clauses called *relative subsumption*
2. the introduction of an inductive mechanism called *relative least general generalization*

The main theoretical result of Plotkin's work was negative since in general there is not a finite relative least general generalization of two clauses. Several attempts have been made to find ways around Plotkin's negative result by restricting the model of the background knowledge and/or the hypotheses language.

This work focuses on the inductive logic programming system called *GOLEM* which implements both approaches, i.e. it restricts the model of the background knowledge as well as the hypotheses language.

GOLEM constructs logic programs — restricted to Horn clauses — from positive and negative examples using information given by the background knowledge. In *GOLEM* the examples and the background knowledge are represented as ground atomic facts. *GOLEM* has a good level of efficiency and is able to learn predicates such as list reverse. quicksort, member, integer multiply and other real-world applications.

The purpose of this work is to present the main conceptual notions used by *GOLEM*. to show in details its ways of operation, to discuss the "quality" of the positive and negative examples in inducing the appropriate expression of the concept, to evidence some of its limitations and show some of its results.

The work is organized as follows: Section 1 highlights some preliminary concepts used in *GOLEM*. Section 2 describes the system, states the algorithm and presents a detailed description of its use in a simple case. Section 3 describes the sort of data needed by *GOLEM* as well as declarations and commands. Section 4 describes in detail the learning process used by *GOLEM* while leaning some concepts. Finally Section 5 presents our conclusions, as well as guidelines for future work.

2 Preliminary Concepts

The Golem system is based on the notions of:

1. ***subsumption***, a relationship of generality between clauses

2. *least general generalization — lgg —* (or anti-unification) of terms, literals and clauses, constructed in terms of θ-subsumption

3. *relative least general generalization — rlgg —* which is the least general generalization in presence of background knowledge

introduced by [Plotkin 71] and presented next.

Definition 2.1 *A clause C_1 is more general than (θ-subsumes) a clause C_2 iff there is a substitution θ such that $C_1\,\theta \subseteq C_2$. The relation C_1 θ-subsumes C_2 is noted by $C_1 \preceq C_2$.*

This definition focuses clauses being regarded as sets of literals related in a disjunctive way. For example, the clause

$$p(X,Y) \leftarrow q(X), r(Z,Y)$$

is considered as the set

$$\{p(X,Y), \neg q(X), \neg r(Z,Y)\}$$

The θ-subsumption relation is a partial ordering of generality over clauses in the absence of background knowledge.

Example 2.1 *Let the clauses*

$C_1 : scene(X) \leftarrow on(X,Y,Z), circle(Y)$

$C_2 : scene(picture1) \leftarrow on(picture1, ring, E), circle(ring), flat(E)$

It can be said that C_1 θ-subsumes C_2, with $\theta = \{picture1/X,\ ring/Y, E/Z\}$

The following properties hold for *θ-subsumption*:

1. if C_1 is more general than C_2 then C_1 logically entails C_2

2. the relation $\preceq$ induces a lattice on the set of all clauses. This means that for any two clauses there is a least upper bound and a greatest lower bound. and both are unique up to equivalence under *θ-subsumption*

The second property leads to the following definition:

Definition 2.2 *The least general generalization — $C = lgg(C_1, C_2)$ — of two clauses C_1 and C_2 is the least upper bound whithin the clause lattice induced by the relation $\preceq$.*

A least general generalization is a generalization which is less general than any other such generalization. This concept is important for concept learning as it forms the basis of cautious generalization algorithms. Cautious generalization assumes that if C_1 and C_2 are true, it is very likely that $lgg(C_1, C_2)$ will also be true [Lavrač 94].

Example 2.2 $lgg(p(g(a), a), p(g(b), b)) = p(g(X), X)$

It should be noted that under the substitution $\{a/X\}$

$$p(g(X), X) \quad \textit{subsumes} \quad p(g(a).a)$$

and that under the substitution $\{b/X\}$

$$p(g(X), X) \quad \textit{subsumes} \quad p(g(b_j.b)$$

Next, three algorithms which compute the lgg of terms, lgg of literals and lgg of clauses respectively are presented [De Raedt 92].

lgg of terms

procedure lgg(t_1 :**term**,t_2 :**term**)
  if $t_1 = t_2$
  then t_1
  else if $t_1 = f(u_1, \ldots, u_k)$ and $t_2 = f(v_1, \ldots, v_k)$
    then $f(lgg(u_1, v_1), \ldots, lgg(u_k, v_k))$
    else variable V, where the same V is used
      everywhere as the lgg of these terms
    endif
  endif
endproc

lgg of literals

procedure lgg(L_1 :**literal**,L_2 :**literal**)
  if $L_1 = sign\ p(t_1, \ldots, t_n)$ and $L_2 = sign\ p(u_1, \ldots, u_n)$
  then $sign\ p(lgg(t_1, u_1), \ldots, lgg(t_n, u_n))$
  else undefined
  endif
endproc

lgg of clauses

procedure lgg(C_1 :**clause**,C_2 :**clause**)
  if $C_1 = \{L_1, \ldots, L_n\}$ and $C_2 = \{K_1, \ldots, K_m\}$
  then $\{U_i \mid U_i = lgg(L_j, K_k)$ for $1 \leq j \leq n$ and $1 \leq k \leq m$ and $lgg(L_j, K_k) \neq undefined\}$
  endif
endproc

The algorithm for constructing the least general generalization — *lgg* — of clauses deals with the problem of constructing a unique clause that generalizes a set of examples.

Although subsumption has been adopted by several learning systems, it has some limitations; in particular, it does not consider the possibility of using the background knowledge eventually available, which could contribute with the process of generalization. Taking into account background knowledge would give the system a notion of semantical generalization. This drawback of lgg is overcome through the use of *relative least general generalization* — *rlgg* — a lgg constructed with respect to the given background knowledge $\mathcal{K}$.

Definition 2.3 *Given the background knowledge consisting of ground facts with $\mathcal{K}$ representing the conjunction of all these facts, the rlgg of two positive training examples e_1 and e_2 relative to the given background knowledge is defined as*

$$rlgg(e_1, e_2) = lgg((e_1 \leftarrow \mathcal{K}), (e_2 \leftarrow \mathcal{K}))$$

The relative least general generalization between 2 examples is defined as the least general generalization between the two corresponding example-clauses: a example-clause has one example as its head and the conjunction $\mathcal{K}$ of all the ground facts which represents the background knowledge as its body. This constitutes the operational definition of rlgg. Section 3.3, pg. 8, presents a detailed construction of the rlgg between two positive instances of the concept *scene*.

3 The ILP System GOLEM

It should be noted that models of a theory (known also as background knowledge $\mathcal{K}$) are not always finite. If the theory contains arbitrary clauses, the model can be infinite — e.g. this happens when the theory contains recursive clauses. Infinite models result in infinite clauses C_1 and C_2 and consequently in an infinite rlgg between them.

In other words, if the logic program P which represents the background knowledge $\mathcal{K}$ consists of arbitrary clauses, the model $\mathcal{M}$ for P can be infinite. In order to deal with situations like that and restrict infinite models to finite ones, the notion of h-easiness was established [Buntine 88] [Muggleton 90] and is discussed in details in [Nicoletti 93]. It has been shown [Muggleton 90] that the Herbrand h-easy model of a special kind of Horn clause program P, known as sintactically generative and defined next, is finite.

Definition 3.1 *The clause $A \leftarrow B_1, B_2, \ldots, B_m$ is syntactically generative whenever the variables in the head A are a subset of the variables in the body $B_1, \ldots, B_m$. The logic program P is syntactically generative iff every clause in P is syntactically generative.*

Thus, if the logic program which defines the domain theory is syntactically generative, the number of ground facts which constitute the model of $\mathcal{K}$ is finite, so the rlgg of a set of those ground facts can be guaranteed to be finite in length.

Although finite, the rlgg's can be very lengthy. An additional way of overcoming Plotkin's results concerning rlgg is through the restriction of the hypotheses language, as sugested in [Frisch 90].

The *GOLEM* system adopts both: it uses h-easiness for restricting the background knowledge and the concept of ij-determinancy for restricting the hypotheses language. It restricts the hypotheses language by restricting [Aha 92]

1. the number of substitutions required in each literal to instantiate existentially quantified variables introduced in the literal
2. the depth of the substitution chains required to instantiate an existentially quantified variable in the clause

The constraints imposed on the hypotheses language[3] by the two bounds — *ij-determinacy* — are a way of controlling the introduction of new variables in the clause being constructed. The existential quantification in *GOLEM* is restricted to being *exists exactly one*. This is otherwise known as Hilbert $\epsilon*$ quantification. Target clauses using non-$\epsilon*$ quantification cannot be learned using *GOLEM* [Dolšak 92].

As stated in [Muggleton 90], in an incremental learning system, learned clauses are added to the background knowledge. So it is reasonable to constrain hypothesized clauses to also being syntactically generative. However, it is important to notice that there are situations where *GOLEM* learns clauses which are not syntactically generative[4].

GOLEM builds constrained hypotheses in a bottom-up approach by incrementally finding the rlggs of examples with respect to the ground background knowledge. If the background knowledge is available in the form of non-ground Horn clauses — *intensional* predicates definitions — the background knowledge has to be transformed into an *h-easy ground model*[5] $\mathcal{M}$.

3.1 Computing the Lgg

Next, an example of interaction with *GOLEM*, for computing the lgg among given atoms is shown. The ending of the input stream of atoms is marked by "**$**" followed by a ".".

```
caiua:/work1/ianomami/IA/carmo/golem/examples>>>../golem
!- rlgg.
g(X,Y).
g(Y,X).
$
```

[3] Referred to as an aspect of *bias* [Utgoff 86].

[4] e.g., when no negative examples are given — see Table 11, pg. 31.

[5] Background knowledge considered as a finite depth Herbrand Model (See [Muggleton 90] [Nicoletti 93].)

```
.
g(A,B).
[Rlgg constructed in 0ms]
!- rlgg.
g(X,Y).
g(X,Y).
$
.
g(A,B).
[Rlgg constructed in 0ms]
!- rlgg.
p(f(X,a),b,c,g(a,b,M)).
p(f(X,e),b,d,g(b,a,M)).
$
.
p(f(A,B),b,C,g(D,E,F)).
[Rlgg constructed in 17ms]
!- rlgg.
p(f(X,a),b,c,g(a,b,M)).
p(f(a,X),e,c,g(X,e,N)).
$
.
p(f(A,B),C,c,g(B,C,D)).
[Rlgg constructed in 0ms]
!- rlgg.
p(M,N,O).
p(f(X),g(Y),3).
p(2,3,4).
p(p(M,N,O),p(M,N,O),p(M,N,O)).
$
.
p(A,B,C).
[Rlgg constructed in 0ms]
!- rlgg.
p(1,1,1,1).
p(□,A,□,B).
p(f(X),B,f(X),A).
$
.
p(A,B,A,C).
[Rlgg constructed in 0ms]
```

Table 1 shows, for various pairs of atoms, the correponding lgg computed by *GOLEM*.

To generate a single clause, *GOLEM* randomly selects a subset of pairs of positive examples[6], generates candidate-rules using rlgg, verifies consistency with the given negative examples (i.e. no negative examples should be covered) and chooses the consistent pair of examples which has the best cover (the greatest number of positive examples covered). This pair is further generalized: a sample of randomly positive

[6]The number of pairs can be established by the user through parameter *settings*. *GOLEM* uses 8 pairs by default.

$Literal_1$	$Literal_2$	$Lgg(Literal_1, Literal_2)$
$g(X,Y)$.	$g(Y,X)$.	$g(A,B)$.
$g(X,Y)$.	$g(X,Y)$.	$g(A,B)$.
$concate([\,],M,N)$.	$concate(X,[\,],N)$.	$concate(A,B,C)$.
$concate([\,],[N\|M],Z)$.	$concate([2],[3\|Y],[2,3\|Y])$.	$concate(A,[B\|C],D)$.
$q([a,b,c,d],X,Y)$.	$q([a,c,d],X,Z)$.	$q([a,A,B\|C],D,E)$.
$q([a,c,d],X,V)$.	$q([a,b,c,d,e],X,V)$.	$q([a,A,B\|C],D,E)$.
$q([1,2],[3,4],[1,2,3,4])$.	$q([a],[b],[a,b])$.	$q([A\|B],[C\|D],[A,E\|F])$.
$q([1,1,2,2,3],$ $[a,b,a],[1,1,2,2,3,a,b,a])$.	$q([a,b],$ $[c],[a,b,c])$.	$q([A,B\|C],$ $[D\|E],[A,B,F\|G])$.
$a([a,b],X,X)$.	$a([a,c,d],V,V)$.	$a([a,A\|B],C,C)$.
$a([a],X,X)$.	$a([b,c],V,V)$.	$a([A\|B],C,C)$.
$p(tree(nil,a,nil))$.	$p(tree(tree(nil,a,nil),a,nil))$.	$p(tree(A,a,nil))$.
$p([1,2,3])$.	$p([4,5,6])$.	$p([A,B,C])$.
$p([1,2,3,43,5,6,7])$.	$p([a])$.	$p([A\|B])$.
$p([\,])$.	$p([1,2,3,4,5,6,7,8])$.	$p(A)$.
$p([1,2,3])$.	$p([a,b,c,d,e])$.	$p([A,B,C\|D])$.
$p([1,1])$.	$p([a,a,b,c,d])$.	$p([A,A\|B])$.
$p([1,1,1,1])$.	$p([2,2,2,3,4,5])$.	$p([A,A,A,B\|C])$.
$p([2,2,2,3,4,5])$.	$p([1,1,1,1])$.	$p([A,A,A,B\|C])$.
$p(f(X,a),b,c,g(a,b,M))$.	$p(f(X,e),b,d,g(b,a,M))$.	$p(f(A,B),b,C,g(D,E,F))$.
$p(f(X,a),b,c,g(a,b,M))$.	$p(f(a,X),e,c,g(X,e,N))$.	$p(f(A,B),C,c,g(B,C,D))$.
$p(A,b,c,D,f(g,h))$.	$p(A,b,v,F,p)$.	$p(A,b,B,C,D)$.
$p(X,Y,Z,p(X,Y,Z))$.	$p(X,Y,Z,p(X,Y,a))$.	$p(A,B,C,p(A,B,D))$.
$p(X,X,Y,p(X,X,Y,a))$.	$p(a,b,m,p(b,a,n,X))$.	$p(A,B,C,p(B,A,D,E))$.
$p([\,],[\,],M,[N\|I],[1,2,3,4\|K],$ $g(M,N,K))$.	$p([\,],a,j,i,[1,2,4,3,7\|K],$ $g(j,1,u))$.	$p([\,],A,B,C,[1,2,D,E\|F],$ $g(B,G,H))$.
$p([1,2],[3,4],[1,2,3,4])$.	$p([a],[b],[a,b])$.	$p([A\|B],[C\|D],[A,E\|F])$.
$p(X,Y,Z)$.	$p(p(X,Y,Z),Y,Y)$.	$p(A,B,C)$.
$p(h(a,b,v,c),h(5,8),M)$.	$p(h(x,y),h(5,8,7,6),B)$.	$p(A,B,C)$.
$p([\,],[\,],[\,],[\,])$.	$p(a,b,a,b)$.	$p(A,B,A,B)$.
$p(1,2,3,4)$.	$p([1,2,3,4],[\,],3,[M\|N])$.	$p(A,B,3,C)$.
$p(1,2,3)$.	$p(2,2,2,2)$.	*[Incompatible predicate symbols]*

Table 1: Lgg between pairs of literals computed by *GOLEM*

examples is selected and their rlgg is computed until the cover of positive examples does not increase with respect to the increase in the sample. When the generalization process ends, the so far obtained clause is then reduced. The reduction process aims to eliminate irrelevant literals in the clause. *GOLEM* performs clauses reduction based on:

1. *functional reduction*, which uses the idea of stating the input and output status of the arguments of a predicate as an additional constraint for constructing the clause (see *mode declaration* next section)

2. *negative-based reduction*, which is based on the notion that a clause should cover as many as possible positive example and none negative example

In order to construct definitions consisting of more than one clause, *GOLEM* uses the covering approach: it constructs a clause covering positive examples, removes the covered examples from the training set and repeats the whole process [Lavrač 94]. *GOLEM* does not automatically create new predicate descriptors.

3.2 *GOLEM* Algorithm

The *GOLEM* algorithm for constructing one clause is given by:

$\mathcal{E}^+$: set of positive examples
$\mathcal{E}^-$: set of negative examples
$\mathcal{M}_h(\mathcal{K})$: h-easy model of background knowledge $\mathcal{K}$
s: a given sample limit
$Pairs(s) = \{ \{e_i, e_j\} \mid e_1, e_j \in \mathcal{E}^+ \}, \mid Pairs(s) \mid = s$
$Lggs = \{ C \text{ consistent wrt } \mathcal{E}^- \mid C = rlgg(\{e, e'\}) \text{ wrt } \mathcal{M}_h(\mathcal{K}), \{e, e'\} \in Pairs(s)\}$
Let $S = \{e, e'\} \in Lggs$ the pair whose *rlgg* has the greatest cover
Do
  Let the random sample $\mathcal{E}^+(s) \subset \mathcal{E}^+, \mid \mathcal{E}^+(s) \mid = s$
  $Lggs = \{ C \text{ consistent wrt } \mathcal{E}^- \mid C = rlgg(S \bigcup \{e'\}) \text{ wrt } \mathcal{M}_h(\mathcal{K}), e' \in Pairs(s)\}$
  Find e' which produces the greatest cover in $Lggs$
  $S = S \bigcup \{e'\}$
  $\mathcal{E}^+ = \mathcal{E}^+ - cover(rlgg(S))$
while increasing cover

After constructing one clause, the positive examples from the set $\mathcal{E}^+$ covered by it are removed, and *GOLEM* repeats the process of clause construction taking into account the positive examples left.

3.3 Learning with *GOLEM* Using only Two Positive Examples

It follows a detailed description of the application of the algorithm where only two positive examples are considered and no negative examples are provided. This example can be found in [Quinlan-91]. In this simple case, the greatest cover will be given by the reduced rlgg of these two examples. Figure 1 shows a picture of the two positive examples considered.

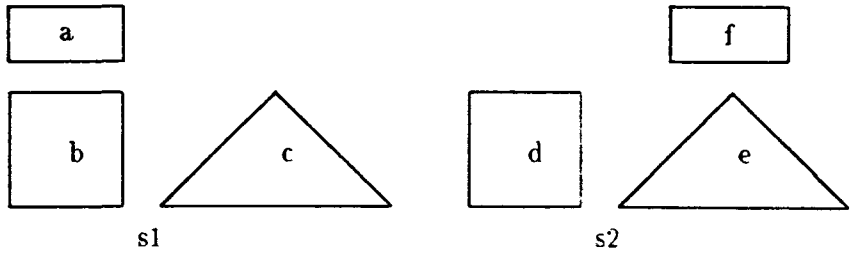

Figure 1: Two positive examples of *scene*

Both examples together with the background knowledge in the format required by *GOLEM* are shown in Table 2.

It follows the computation of the rlgg of both examples.

$$\text{rlgg(s1,s2)} \quad = \quad \text{lgg((scene(s1)} \leftarrow \mathcal{K}\text{),(scene(s2)} \leftarrow \mathcal{K}\text{))}$$

Examples	Background Knowledge ($\mathcal{K}$)
scene(s1) $\oplus$	on(s1,a,b)
scene(s2) $\oplus$	on(s2,f,e)
	left_of(s1,b,c)
	left_of(s2,d,e)
	rectangle(a)
	rectangle(f)
	square(b)
	square(d)
	triangle(c)
	triangle(e)

Table 2: Positive examples and Background Knowledge for *scene*

where

$$\begin{aligned} \text{scene(s1)} \leftarrow \mathcal{K} \equiv \quad & \neg\ \text{on(s1,a,b)} \vee \neg\ \text{on(s2,f,e)} \\ & \vee \neg\ \text{left\_of(s1,b,c)} \vee \neg\ \text{left\_of(s2,d,e)} \\ & \vee \neg\ \text{rectangle(a)} \vee \neg\ \text{rectangle(f)} \\ & \vee \neg\ \text{square(b)} \vee \neg\ \text{square(d)} \\ & \vee \neg\ \text{triangle(c)} \vee \neg\ \text{triangle(e)} \\ & \vee \text{scene(s1)} \end{aligned}$$

$$\begin{aligned} \text{scene(s2)} \leftarrow \mathcal{K} \equiv \quad & \neg\ \text{on(s1,a,b)} \vee \neg\ \text{on(s2,f,e)} \\ & \vee \neg\ \text{left\_of(s1,b,c)} \vee \neg\ \text{left\_of(s2,d,e)} \\ & \vee \neg\ \text{rectangle(a)} \vee \neg\ \text{rectangle(f)} \\ & \vee \neg\ \text{square(b)} \vee \neg\ \text{square(d)} \\ & \vee \neg\ \text{triangle(c)} \vee \neg\ \text{triangle(e)} \\ & \vee \text{scene(s2)} \end{aligned}$$

The following tables show the computation of the lgg between literals as well as the resulting variables bindings, to obtain the rlgg(s1,s2).

—lgg of—	—result—
¬ on(s1,a,b) , ¬ on(s1,a,b)	¬ on(s1,a,b)
¬ on(s1,a,b) , ¬ on(s2,f,e)	¬ on(X,Y,Z)
¬ on(s1,a,b) , ¬ left_of(s1,b,c)	undefined
¬ on(s1,a,b) , ¬ left_of(s2,d,e)	undefined
¬ on(s1,a,b) , ¬ rectangle(a)	undefined
¬ on(s1,a,b) , ¬ rectangle(f)	undefined
¬ on(s1,a,b) , ¬ square(b)	undefined
¬ on(s1,a,b) , ¬ square(d)	undefined
¬ on(s1,a,b) , ¬ triangle(c)	undefined
¬ on(s1,a,b) , ¬ triangle(e)	undefined
¬ on(s1,a,b) , scene(s2)	undefined

—Bindings—
X = lgg(s1,s2), Y = lgg(a,f), Z = lgg(b,e)

—lgg of—	—result—
¬ on(s2,f,e) , ¬ on(s1,a,b)	¬ on(X,Y,Z)
¬ on(s2,f,e) , ¬ on(s2,f,e)	¬ on(s2,f,e)
¬ on(s2,f,e) , ¬ left_of(s1,b,c)	undefined
¬ on(s2,f,e) , ¬ left_of(s2,d,e)	undefined
¬ on(s2,f,e) , ¬ rectangle(a)	undefined
¬ on(s2,f,e) , ¬ rectangle(f)	undefined
¬ on(s2,f,e) , ¬ square(b)	undefined
¬ on(s2,f,e) , ¬ square(d)	undefined
¬ on(s2,f,e) , ¬ triangle(c)	undefined
¬ on(s2,f,e) , ¬ triangle(e)	undefined
¬ on(s2,f,e) , scene(s2)	undefined
—Bindings—	
X1 = lgg(s2,s1), Y1 = lgg(f,a), Z1 = lgg(e,b)	

It is important to notice that the lgg of two literals is not commutative. i.e.

$$lgg(s1,s2) \neq lgg(s2,s1)$$

—lgg of—	—result—
¬ left_of(s1,b,c) , ¬ on(s1,a,b)	undefined
¬ left_of(s1,b,c) , ¬ on(s2,f,e)	undefined
¬ left_of(s1,b,c) , ¬ left_of(s1,b,c)	¬ left_of(s1,b,c)
¬ left_of(s1,b,c) , ¬ left_of(s2,d,e)	¬ left_of(X,M,N)
¬ left_of(s1,b,c) , ¬ rectangle(a)	undefined
¬ left_of(s1,b,c) , ¬ rectangle(f)	undefined
¬ left_of(s1,b,c) , ¬ square(b)	undefined
¬ left_of(s1,b,c) , ¬ square(d)	undefined
¬ left_of(s1,b,c) , ¬ triangle(c)	undefined
¬ left_of(s1,b,c) , ¬ triangle(e)	undefined
¬ left_of(s1,b,c) , scene(s2)	undefined
—Bindings—	
X = lgg(s1,s2), M = lgg(b,d), N = lgg(c,e)	

—lgg of—	—result—
¬ left_of(s2,d,e) , ¬ on(s1,a,b)	undefined
¬ left_of(s2,d,e) , ¬ on(s2,f,e)	undefined
¬ left_of(s2,d,e) , ¬ left_of(s1,b,c)	¬ left_of(X,M,N)
¬ left_of(s2,d,e) , ¬ left_of(s2,d,e)	¬ left_of(s2,d,e)
¬ left_of(s2,d,e) , ¬ rectangle(a)	undefined
¬ left_of(s2,d,e) , ¬ rectangle(f)	undefined
¬ left_of(s2,d,e) , ¬ square(b)	undefined
¬ left_of(s2,d,e) , ¬ square(d)	undefined
¬ left_of(s2,d,e) , ¬ triangle(c)	undefined
¬ left_of(s2,d,e) , ¬ triangle(e)	undefined
¬ left_of(s2,d,e) , scene(s2)	undefined
—Bindings—	
X1 = lgg(s2,s1), M1 = lgg(d,b), N1 = lgg(e,c)	

—lgg of—	—result—
¬ rectangle(a) . ¬ on(s1,a,b)	undefined
¬ rectangle(a) . ¬ on(s2,f,e)	undefined
¬ rectangle(a) . ¬ left_of(s1,b,c)	undefined
¬ rectangle(a) . ¬ left_of(s2,d,e)	undefined
¬ rectangle(a) . ¬ rectangle(a)	¬ rectangle(a)
¬ rectangle(a) . ¬ rectangle(f)	¬ rectangle(Y)
¬ rectangle(a) . ¬ square(b)	undefined
¬ rectangle(a) . ¬ square(d)	undefined
¬ rectangle(a) . ¬ triangle(c)	undefined
¬ rectangle(a) . ¬ triangle(e)	undefined
¬ rectangle(a) . scene(s2)	undefined

—Bindings—
Y = lgg(a,f)

—lgg of—	—result—
¬ rectangle(f) . ¬ on(s1,a,b)	undefined
¬ rectangle(f) . ¬ on(s2,f,e)	undefined
¬ rectangle(f) . ¬ left_of(s1,b,c)	undefined
¬ rectangle(f) . ¬ left_of(s2,d,e)	undefined
¬ rectangle(f) . ¬ rectangle(a)	¬ rectangle(Y)
¬ rectangle(f) . ¬ rectangle(f)	¬ rectangle(f)
¬ rectangle(f) . ¬ square(b)	undefined
¬ rectangle(f) . ¬ square(d)	undefined
¬ rectangle(f) . ¬ triangle(c)	undefined
¬ rectangle(f) . ¬ triangle(e)	undefined
¬ rectangle(f) . scene(s2)	undefined

—Bindings—
Y1 = lgg(f,a)

—lgg of—	—result—
¬ square(b) . ¬ on(s1,a,b)	undefined
¬ square(b) . ¬ on(s2,f,e)	undefined
¬ square(b) . ¬ left_of(s1,b,c)	undefined
¬ square(b) . ¬ left_of(s2,d,e)	undefined
¬ square(b) . ¬ rectangle(a)	undefined
¬ square(b) . ¬ rectangle(f)	undefined
¬ square(b) . ¬ square(b)	¬ square(b)
¬ square(b) . ¬ square(d)	¬ square(M)
¬ square(b) . ¬ triangle(c)	undefined
¬ square(b) . ¬ triangle(e)	undefined
¬ square(b) . scene(s2)	undefined

—Bindings—
M = lgg(b,d)

—lgg of—	—result—
¬ square(d) , ¬ on(s1,a,b)	undefined
¬ square(d) , ¬ on(s2,f,e)	undefined
¬ square(d) , ¬ left_of(s1,b,c)	undefined
¬ square(d) , ¬ left_of(s2,d,e)	undefined
¬ square(d) , ¬ rectangle(a)	undefined
¬ square(d) , ¬ rectangle(f)	undefined
¬ square(d) , ¬ square(b)	¬ square(M)
¬ square(d) , ¬ square(d)	¬ square(d)
¬ square(d) , ¬ triangle(c)	undefined
¬ square(d) , ¬ triangle(e)	undefined
¬ square(d) , scene(s2)	undefined

—Bindings—
M1 = lgg(d,b)

—lgg of—	—result—
¬ triangle(c) , ¬ on(s1,a,b)	undefined
¬ triangle(c) , ¬ on(s2,f,e)	undefined
¬ triangle(c) , ¬ left_of(s1,b,c)	undefined
¬ triangle(c) , ¬ left_of(s2,d,e)	undefined
¬ triangle(c) , ¬ rectangle(a)	undefined
¬ triangle(c) , ¬ rectangle(f)	undefined
¬ triangle(c) , ¬ square(b)	undefined
¬ triangle(c) , ¬ square(d)	undefined
¬ triangle(c) , ¬ triangle(c)	¬ triangle(c)
¬ triangle(c) , ¬ triangle(e)	¬ triangle(N)
¬ triangle(c) , scene(s2)	undefined

—Bindings—
N = lgg(c,e)

—lgg of—	—result—
¬ triangle(e) , ¬ on(s1,a,b)	undefined
¬ triangle(e) , ¬ on(s2,f,e)	undefined
¬ triangle(e) , ¬ left_of(s1,b,c)	undefined
¬ triangle(e) , ¬ left_of(s2,d,e)	undefined
¬ triangle(e) , ¬ rectangle(a)	undefined
¬ triangle(e) , ¬ rectangle(f)	undefined
¬ triangle(e) , ¬ square(b)	undefined
¬ triangle(e) , ¬ square(d)	undefined
¬ triangle(e) , ¬ triangle(c)	¬ triangle(N)
¬ triangle(e) , ¬ triangle(e)	¬ triangle(e)
¬ triangle(e) , scene(s2)	undefined

—Bindings—
N1 = lgg(e,c)

—lgg of—	—result—
¬ scene(s1) , ¬ on(s1,a,b)	undefined
¬ scene(s1) , ¬ on(s2,f,e)	undefined
¬ scene(s1) , ¬ left_of(s1,b,c)	undefined
¬ scene(s1) , ¬ left_of(s2,d,e)	undefined
¬ scene(s1) , ¬ rectangle(a)	undefined
¬ scene(s1) , ¬ rectangle(f)	undefined
¬ scene(s1) , ¬ square(b)	undefined
¬ scene(s1) , ¬ square(d)	undefined
¬ scene(s1) , ¬ triangle(c)	undefined
¬ scene(s1) , ¬ triangle(e)	undefined
¬ scene(s1) , scene(s2)	scene(X)
—Bindings— X = lgg(s1,s2)	

So,

$$\text{lgg}((\text{scene}(s1) \leftarrow \mathcal{K}),(\text{scene}(s2) \leftarrow \mathcal{K}))$$

which can be written as

¬ on(s1,a,b) ∨ ¬ on(X,Y,Z) ∨¬ on(s2,f,e) ∨¬ on(X1,Y1,Z1)
∨ ¬ left_of(s1,b,c) ∨ ¬ left_of(X,M,N) ∨ ¬ left_of(X1,M1,N1) ∨ ¬ left_of(s2,d,e)
∨ ¬ rectangle(a) ∨ ¬ rectangle(Y) ∨ ¬ rectangle(Y1)∨ ¬ rectangle(f)
∨ ¬ square(b) ∨ ¬ square(M) ∨ ¬ square(M1) ∨ ¬ square(d)
∨ ¬ triangle(c) ∨ ¬ triangle(N) ∨ ¬ triangle(N1) ∨ ¬ triangle(e)
∨ scene(X)

By taking off irrelevant literals which are true since they are in the background knowledge, the clause can be further reduced to

¬ on(X,Y,Z) ∨¬ on(X1,Y1,Z1)
¬ left_of(X,M,N) ∨ ¬ left_of(X1,M1,N1)
¬ rectangle(Y) ∨ ¬ rectangle(Y1)
¬ square(M) ∨ ¬ square(M1)
¬ triangle(N) ∨ ¬ triangle(N1)
∨ scene(X)

Since the literals

on(X1,Y1,Z1)
left_of(X1,M1,N1)
rectangle(Y1)
square(M1)
triangle(N1)

have variables which do not appear in the head of the clause — scene(X) — as well as in any previous literals, they can be eliminated, and the clause is reduced to

¬ on(X,Y,Z) ∨
¬ left_of(X,M,N) ∨
¬ rectangle(Y) ∨
¬ square(M) ∨
¬ triangle(N) ∨
scene(X)

which is further reduced to scene(X)

This last sort of reduction will always happen in the absence of negative examples, since *GOLEM* eliminates literals from the body of the clause which do not cause the clause to cover negative examples.

4 Data and Commands used by *GOLEM*

As explained before, *GOLEM*[7] is a first order induction algorithm which can generate rules from a given set of examples, where

- *example* is a first order ground atom
- *rule* is a first order Horn clause

Rules can be used to classify new examples. The syntax of atoms and rules (or clauses) — see Table 3 — follows that of Edinburgh Prolog (without operator definition and queries).

structure	format	comments
ground atom	$predicate(arg_1, arg_2, \ldots, arg_n)$. e.g. $likes(maria, mother(maria))$.	$arg_1, arg_2, \ldots, arg_n$ are terms, possibly with function symbols and no variables
clause (or rule)	$Head :- A_1, A_2, \ldots, A_n$ e.g. $member(A,[B,C\|D]) :- natural(B), member(A,[C\|D])$.	$Head, A_1, A_2, \ldots, A_n$ are also atoms, but containing at least one variable

Table 3: Syntax of atoms and rules in *GOLEM*

[7]It is implemented in Sun's implemetation of C and runs under SunOS Release 4.0. The implementation described here is due to Cao Feng and Stephen Muggleton [Feng 90].

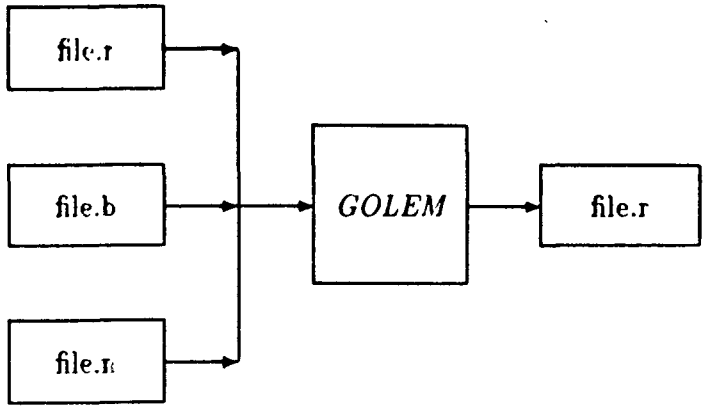

Figure 2: Data files used by *GOLEM*

4.1 Files

GOLEM needs three types of input data files and produce (depending on user's choice) an output file as shown in Figure 2. The type and extention of the necessary files, as well as its contents are described in Table 4.

type	extention	comments
foreground	.*f*	set of positives examples. Each element is a ground atom implied by the target predicate
background	.*b*	set of examples which are believed to be relevant to the example in the foreground set It constains, as well, the declarations part which defines the modes and determinations of each predicate
negative	.*n*	set of ground atoms that share the same predicate symbol as the elements of the foreground set, but are false with respect to the target predicate
rules	.*r*	set of input or output rules

Table 4: Types of data files used by *GOLEM*

GOLEM can be used in two different modes of operation, batch and interactive, as described in Table 5.

4.2 Mode and Determination Declaration

Commands in *GOLEM* can be declared in the background example file, including *mode* and *determination* declaration [Feng 90] — see Table 6.

Through a *mode* declaration it is possible to specify a relationship between the ar-

mode	comments
batch	Golem accepts data files and produces a set of rules. At least one foreground, background and negative files must be present
interactive	Golem interacts with the user to perform some specified induction task

Table 5: Modes of operation in *GOLEM*

declaration	sintax
mode	*mode*(*predicate*(+,+,...,−,−)). where: + stands for input − stands for output with respect to *predicate*
determination	*determination*(*predicate1*(_,_,...,_), *predicate2*(_,_,...,_)). where: *predicate1* is the head of the expected rules *predicate2* is a predicate found in the foreground or background examples

Table 6: Declarations in *GOLEM*

guments of a predicate; the values of the arguments being declared output can be computed from the values of the arguments being declared input. When there are no mode declarations for some predicate, by default it is assumed that all arguments are input.

Through a *determination* declaration it is possible to specify a relationship between the predicate in the head of the expected rules and the other predicates found in the background examples. A determination declaration is a way of telling *GOLEM* that some predicates should be used to construct the expected rule(s). When there are no determination declarations for some predicates, by default it is assumed that the foreground and all background predicates can be used to construct a rule.

4.3 Other Commands

GOLEM also provides commands which can be used to get help, to display memory usage, for debugging and tracing information, system statistics, I/O, system setting parameters, etc. Some of these commands are presented in the order of their likely usage [Feng 90].

show(Arg1). display examples in memory. Arg1 can be any of *fore*, *back*, *neg*, *rules* or *hypothesis* respectively for displaying positive examples, background knowledge, negative examples, rules in memory or the current hypothesis

read(Arg1,Arg2). read in files with the specified file name. Arg1 can be any of *fore*,

back, *neg*, *rule* or *all* respectively for positive examples, background knowledge, negative examples, rules in memory or to read in all the existing files with the file name stem specified by Arg2

induce. induce rules

doall(Arg1). read in files with the specified file name stem Arg1, induce rules and write them to the file with extension .r

addhyp. add reduced hypothesis to current rule set in memory

write(Arg1,Arg2). output rules and existing examples to the respective files. Arg1 and Arg2 are similar to those in read(Arg1,Arg2)

subduce. compute the number of positive examples covered by the current hypothesis

subducen. compute the number of negative examples covered by the current hypothesis

settings. display all system settings described below, together with its default values

settings	default value	description
i	2	*i – determinacy*
j	2	*j – determinacy*
noise	0	limit of the permissible number of negative examples being misclassified by each rule
rlggsample	8	limit of a sample of example pairs used to construct clauses
testsample	500	limite of a sample of negative examples used to test clauses

set(Arg1,Arg2). reset any of the settings described above. Arg1 is any of the settings and Arg2 is a number

covers. compute the number of positive examples covered by rules in memory

coversn. compute the number of negative examples covered by rules in memory

5 Using *GOLEM*

In this section some examples of the learning process carried out by *GOLEM* are presented in detail.

5.1 Learning the Concept *class*

This example is due to Feng and Muggleton [Feng 90]. The ten positive and nine negative examples used for learning the *class* concept are given in Table 7 and the background knowledge in Table 8.

Examples	
Positive	Negative
(1) class(a1,mammal).	class(a1,reptile).
(2) class(a2,mammal).	class(a5,mammal).
(3) class(a3,mammal).	class(a5,reptile).
(4) class(a4,mammal).	class(a5,bird).
(5) class(a5,fish).	class(a6,mammal).
(6) class(a6,reptile).	class(a7,mammal).
(7) class(a7,reptile).	class(a8,mammal).
(8) class(a8,bird).	class(a9,mammal).
(9) class(a9,bird).	class(a10,mammal).
(10) class(a10,amphibian).	

Table 7: Positive and negative examples for learning *class*

Background Knowledge (K)		
has_covering(a1,hair).	milk(a1,t).	habitat(a1,land).
has_covering(a2,none).	milk(a2,f).	habitat(a2,sea).
has_covering(a3,hair).	milk(a3,t).	habitat(a3,sea).
has_covering(a4,hair).	milk(a4,f).	habitat(a4,air).
has_covering(a5,scales).	milk(a5,f).	habitat(a5,sea).
has_covering(a6,scales).	milk(a6,f).	habitat(a6,land).
has_covering(a7,scales).	milk(a7,f).	habitat(a7,sea).
has_covering(a8,feathers).	milk(a8,f).	habitat(a8,air).
has_covering(a9,feathers).	milk(a9,f).	habitat(a9,land).
has_covering(a10,none).	milk(a10,f).	habitat(a10,land).
homeothermic(a1,t).	eggs(a1,f).	gills(a1,f).
homeothermic(a2,t).	eggs(a2,f).	gills(a2,f).
homeothermic(a3,t).	eggs(a3,f).	gills(a3,f).
homeothermic(a4,t).	eggs(a4,f).	gills(a4,f).
homeothermic(a5,f).	eggs(a5,t).	gills(a5,t).
homeothermic(a6,f).	eggs(a6,t).	gills(a6,f).
homeothermic(a7,f).	eggs(a7,t).	gills(a7,f).
homeothermic(a8,t).	eggs(a8,t).	gills(a8,f).
homeothermic(a9,t).	eggs(a9,t).	gills(a9,f).
homeothermic(a10,f).	eggs(a10,t).	gills(a10,f).

Table 8: Background Knowledge for learning *class*

With this sort of data *GOLEM* learns the following concept of *class*

```
class(a5,fish).
class(a10,amphibian).
class(A,mammal) :- milk(A,t).
class(A,bird) :- has_covering(A,feathers).
class(A,reptile) :- has_covering(A,scales), gills(A,f).
```

It follows a listing of an interactive session with *GOLEM* while learning this concept. Except for the comments included to explain the commands used during this session, the complete output from the interactive process is presented. The input files used are:

animals.f	positive examples
animals.n	negative examples
animals.b	background knowledge

First of all, *GOLEM* is loaded using option -d (`golem -d`) to display tracing and debugging information.

```
!- ianomami:/work1/ianomami/IA/carmo/golem/examples>>>../golem -d
[Debugging ON]
```

All positive, negative examples and background knowledge are read into memory through the command `read(all,animals).`

```
!- read(all,animals).
[Read in 166ms]
```

Positive examples in memory are shown by the following command

```
!- show(fore).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a6,reptile).
class(a7,reptile).
class(a8,bird).
class(a9,bird).
class(a10,amphibian).
```

Background knowledge in memory is shown by the `show(back).` command. Note that positive examples are automatically included by *GOLEM* as background knowledge as well.

```
!- show(back).
has_covering(a1,hair).
has_covering(a2,none).
has_covering(a3,hair).
```

```
has_covering(a4,hair).
has_covering(a5,scales).
has_covering(a6,scales).
has_covering(a7,scales).
has_covering(a8,feathers).
has_covering(a9,feathers).
has_covering(a10,none).
milk(a1,t).
milk(a2,t).
milk(a3,t).
milk(a4,t).
milk(a5,f).
milk(a6,f).
milk(a7,f).
milk(a8,f).
milk(a9,f).
milk(a10,f).
homeothermic(a1,t).
homeothermic(a2,t).
homeothermic(a3,t).
homeothermic(a4,t).
homeothermic(a5,f).
homeothermic(a6,f).
homeothermic(a7,f).
homeothermic(a8,t).
homeothermic(a9,t).
homeothermic(a10,f).
habitat(a1,land).
habitat(a2,sea).
habitat(a3,sea).
habitat(a4,air).
habitat(a5,sea).
habitat(a6,land).
habitat(a7,sea).
habitat(a8,air).
habitat(a9,land).
habitat(a10,land).
eggs(a1,f).
eggs(a2,f).
eggs(a3,t).
eggs(a4,f).
eggs(a5,t).
eggs(a6,t).
eggs(a7,t).
eggs(a8,t).
eggs(a9,t).
eggs(a10,t).
gills(a1,f).
gills(a2,f).
gills(a3,f).
gills(a4,f).
gills(a5,t).
```

```
gills(a6,f).
gills(a7,f).
gills(a8,f).
gills(a9,f).
gills(a10,f).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a6,reptile).
class(a7,reptile).
class(a8,bird).
class(a9,bird).
class(a10,amphibian).
```

Negative examples in memory are shown by the following command

```
!- show(neg).
class(a1,reptile).
class(a5,mammal).
class(a5,reptile).
class(a5,bird).
class(a6,mammal).
class(a7,mammal).
class(a8,mammal).
class(a9,mammal).
class(a10,mammal).
```

Rule induction is activated by the command **induce.** as shown below. It should be noted that defaults settings are being used. This means that *GOLEM* is working with 22-determinancy, noise is not allowed (noise = 0) meaning that no negative examples can be covered by the induced rules, and the sample size used to compute the rlgg is 8. In this case the 8 pairs of positive examples considered by *GOLEM* are: {(2,9), (2,10), (3,7), (4,5), (4,8), (4,9), (5,7), (6,7)}

```
!- induce.
[Pair:]
class(a2,mammal).
class(a9,bird).
[Pair:]
class(a2,mammal).
class(a10,amphibian).
[Pair:]
class(a3,mammal).
class(a7,reptile).
[Pair:]
class(a4,mammal).
class(a5,fish).
[Pair:]
class(a4,mammal).
class(a8,bird).
[Pair:]
```

```
class(a4,mammal).
class(a9,bird).
[Pair:]
class(a5,fish).
class(a7,reptile).
[Pair:]
class(a6,reptile).
class(a7,reptile).
```

At this point, after having calculated the rlgg of these 8 pairs of examples, *GOLEM* chooses the one which has the greatest coverage (the rlgg obtained from examples 6 and 7 in this particular case — see below). This rlgg covers 2 positive examples leaving out 8 potential good examples from the 10 positive examples in memory. Next, *GOLEM* computes the rlgg of these two examples (6 and 7) with a third one chosen from the potential good examples left. It does that eight times (rlggsample is 8 by default), randomly chosing each time the third example from the potential good examples left. Unfortunately the tracing facility implemented in *GOLEM* does not allow to identify the randomly chosen third example.

```
[Pos-neg cover=2,potential-good-examples=8]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,F), gills(A,f).
[Number of negatives covered=4]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,F), gills(A,f).
[Number of negatives covered=4]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=3]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,F), gills(A,f).
[Number of negatives covered=4]
[Overgeneral]
[Rlgg of:]
```

```
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,scales), milk(A,f), homeothermic(A,
f), habitat(A,C), eggs(A,t), gills(A,D), gills(a5,t).
[Number of negatives covered=5]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,f), homeothermic(A,D),
 habitat(A,E), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=5]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,f), homeothermic(A,D),
 habitat(A,E), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=5]
[Overgeneral]
[Rlgg of:]
class(a6,reptile).
class(a7,reptile).
  [is]
class(A,B) :- has_covering(A,C), milk(A,f), homeothermic(A,f),
 habitat(A,D), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=3]
[Overgeneral]
```

After constructing the sample of 8 rlggs of the examples 6 and 7 with a third example, *GOLEM* chooses the one with the greatest coverage and reduces the clause, showing as well the time taken to construct the hypothesis (764ms in this case).

```
[Cover=2]
[Reducing clause]
class(A,reptile) :- has_covering(A,scales), gills(A,f).
[Hypothesis constructed in 764ms]
```

The **subduce.** and **subducen.** commands can be used to compute respectively the number of positive and negative examples covered by the current hypothesis.

```
!- subduce.
class(a6,reptile).
class(a7,reptile).
[Positive Cover = 2]
[Cover found in 0ms]
!- subducen.
[Negative Cover = 0]
[Negative cover found in 0ms]
```

The reduced hypothesis can be added to the current rule set in memory by the command **addhyp**. which also removes from the set of positive examples in memory, the examples — **class(a6,reptile)** and **classe(a7,reptile)** in this case — covered by this new added rule.

```
!- addhyp.
[REMOVED: 2 FORES, 0 NEGS]
[Foreground cover removed in 0ms]
!- show(fore).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a8,bird).
class(a9,bird).
class(a10,amphibian).
!- show(rules).
class(A,reptile) :- has_covering(A,scales), gills(A,f).
```

The **induce.** command below repeats the process using the 8 positive examples in memory which have not yet been covered. It constructs the rlgg of the following 8 pairs of examples {(2,9), (3,5), (3,10), (4,5), (4,10), (5,10), (5,9), (8,10)}

```
!- induce.
[Pair:]
class(a2,mammal).
class(a9,bird).
[Pair:]
class(a3,mammal).
class(a5,fish).
[Pair:]
class(a3,mammal).
class(a10,amphibian).
[Pair:]
class(a4,mammal).
class(a5,fish).
[Pair:]
class(a4,mammal).
class(a10,amphibian).
[Pair:]
class(a5,fish).
class(a10,amphibian).
[Pair:]
class(a8,bird).
class(a9,bird).
[Pair:]
class(a8,bird).
class(a10,amphibian).
```

Again, the two examples with the greatest rlgg coverage are chosen. They are examples 8 and 9 in this particular case. Next the process of constructing the rlgg of these examples plus one randomly chosen is repeated.

```
[Pos-neg cover=2,potential-good-examples=6]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,t),
 habitat(A,E), eggs(A,F), gills(A,f), gills(a5,t).
[Number of negatives covered=3]
[Overgeneral]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,t),
 habitat(A,E), eggs(A,F), gills(A,f), gills(a5,t).
[Number of negatives covered=3]
[Overgeneral]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,t),
 habitat(A,E), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=2]
[Overgeneral]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,t),
 habitat(A,E), eggs(A,F), gills(A,f), gills(a5,t).
[Number of negatives covered=3]
[Overgeneral]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,f), homeothermic(A,D),
 habitat(A,E), eggs(A,t), gills(A,F), gills(a5,t).
[Number of negatives covered=8]
[Overgeneral]
[Rlgg of:]
class(a8,bird).
class(a9,bird).
  [is]
class(A,B) :- has_covering(A,C), milk(A,f), homeothermic(A,D),
 habitat(A,E), eggs(A,t), gills(A,f), gills(a5,t).
[Number of negatives covered=5]
[Overgeneral]
```

Again the rlgg with the greatest coverage is chosen and the clause is reduced to a new hypothesis. It is important to notice the difference between hypothesis and rules. A

hypothesis is converted to a rule only after the execution of the command **addhyp**. which also removes the positive examples covered by the new rule.

```
[Cover=2]
[Reducing clause]
class(A,bird) :- has_covering(A,feathers).
[Hypothesis constructed in 681ms]
!- show(hypothesis).
class(A,bird) :- has_covering(A,feathers).
!- show(rules).
class(A,reptile) :- has_covering(A,scales), gills(A,f).
!- show(fore).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a8,bird).
class(a9,bird).
class(a10,amphibian).
!- addhyp.
[REMOVED: 2 FORES, 0 NEGS]
[Foreground cover removed in 17ms]
!- show(fore).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a10,amphibian).
!- show(rules).
class(A,bird) :- has_covering(A,feathers).
class(A,reptile) :- has_covering(A,scales), gills(A,f).
```

The inductive process is again repeated — using the 6 positive examples left — constructing the rlgg of the following 8 pairs of examples {(1,2), (1,3), (1,4), (2,5), (2,10), (3,10), (4,5), (4,10)}

```
!- induce.
[Pair:]
class(a1,mammal).
class(a2,mammal).
[Pair:]
class(a1,mammal).
class(a3,mammal).
[Pair:]
class(a1,mammal).
class(a4,mammal).
[Pair:]
class(a2,mammal).
class(a5,fish).
[Pair:]
class(a2,mammal).
```

```
class(a10,amphibian).
[Pair:]
class(a3,mammal).
class(a10,amphibian).
[Pair:]
class(a4,mammal).
class(a5,fish).
[Pair:]
class(a4,mammal).
class(a10,amphibian).
```

The examples with the greatest rlgg coverage are 1 and 2. Now the rlgg of these two examples and a third one is constructed.

```
[Pos-neg cover=3,potential-good-examples=3]
[Rlgg of:]
class(a1,mammal).
class(a2,mammal).
  [is]
class(A,mammal) :- has_covering(A,B), milk(A,t), homeothermic(A,
t), habitat(A,C), eggs(A,D), gills(A,f), gills(a5,t).
[Number of negatives covered=0]
[OK]
[Rlgg of:]
class(a1,mammal).
class(a2,mammal).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,F), gills(A,F).
[Number of negatives covered=4]
[Overgeneral]
[Rlgg of:]
class(a1,mammal).
class(a2,mammal).
  [is]
class(A,B) :- has_covering(A,C), milk(A,D), homeothermic(A,D),
 habitat(A,E), eggs(A,F), gills(A,f).
[Number of negatives covered=4]
[Overgeneral]
[Best-atom class(a3,mammal).]
```

Now *GOLEM* constructs the new hypothesis which covers 4 examples and does not leave any potential good examples out.

```
[Pos-neg cover=4,potential-good-examples=0]
[Cover=4]
[Reducing clause]
class(A,mammal) :- milk(A,t).
[Hypothesis constructed in 531ms]
!- show(hypothesis).
class(A,mammal) :- milk(A,t).
!- show(rules).
class(A,bird) :- has_covering(A,feathers).
```

```
class(A,reptile) :- has_covering(A,scales), gills(A,f).
!- show(fore).
class(a1,mammal).
class(a2,mammal).
class(a3,mammal).
class(a4,mammal).
class(a5,fish).
class(a10,amphibian).
```

This new hypothesis is added to the current rule set leaving two positive examples out. for which no good hypothesis can be found.

```
!- addhyp.
[REMOVED: 4 FORES, 0 NEGS]
[Foreground cover removed in 17ms]
!- show(rules).
class(A,mammal) :- milk(A,t).
class(A,bird) :- has_covering(A,feathers).
class(A,reptile) :- has_covering(A,scales), gills(A,f).
!- show(fore).
class(a5,fish).
class(a10,amphibian).
!- induce.
[Pair:]
class(a5,fish).
class(a10,amphibian).
[No good hypothesis found]
```

It can be seen that the three rules discovered by *GOLEM* do not cover negative examples (since the settings has noise = 0) and cover 8 positive examples, leaving 2 out.

If *GOLEM* is used through the `doall(animals).` command, the obtained rules are automaticaly written in the file *.r* as well as the positive examples not eventually covered by the constructed rules.

5.2 Learning the Concept *scene*

This example is concerned with learning the concept of a scene from positive and negative examples. Each scene consists of three different objects — chosen among rectangle, square, triangle and line. The five positive and five negative examples are illustrated in Figures 3 and 4 respectively. where it can be seen that a scene can be described as

a rectangle on any other object and
a square on the left of any other object

The expression of Figures 3 and 4 as positive and negative examples as well as background knowledge to be used by *GOLEM* are shown in Tables 9 and 10.

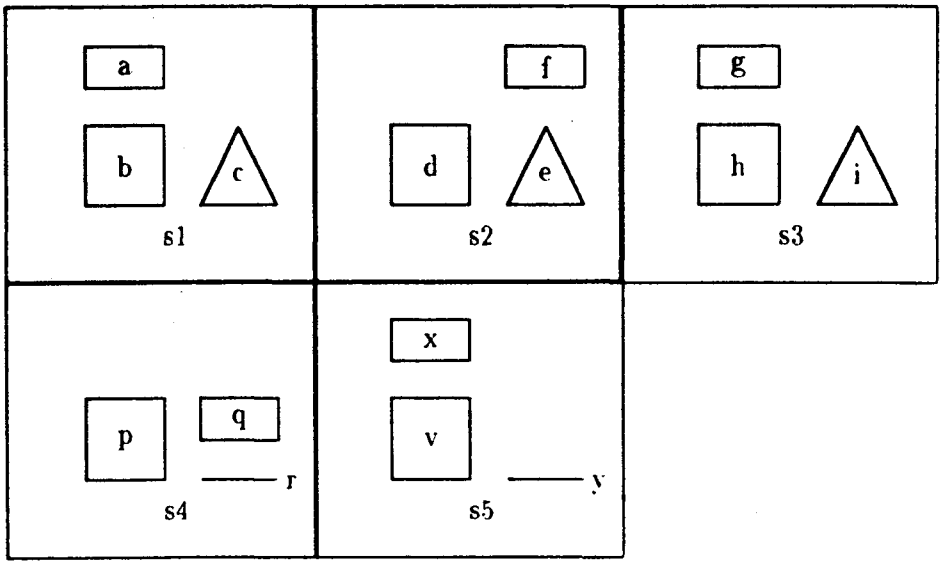

Figure 3: Positive examples for learning *scene*

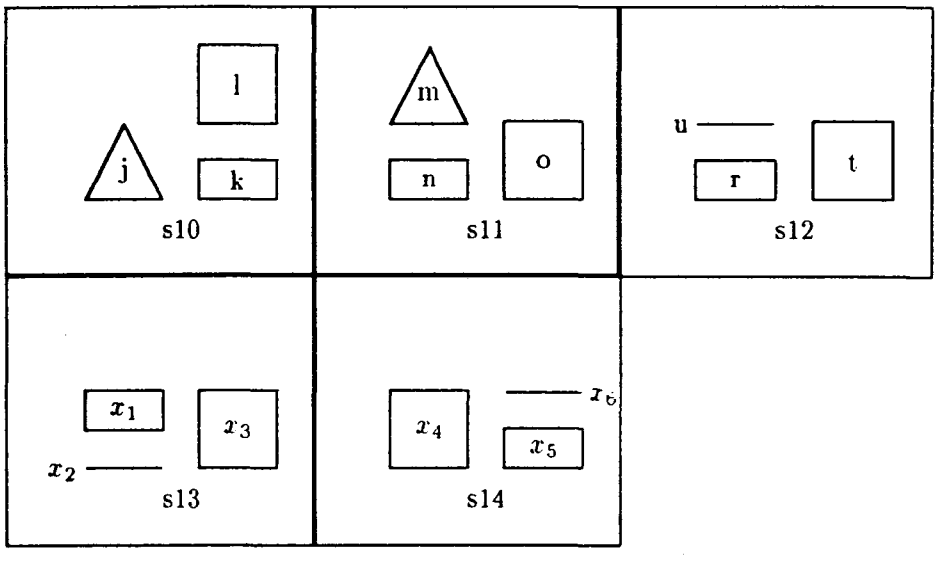

Figure 4: Negative examples for learning *scene*

Positive Examples	Negative Examples
scene(s1).	scene(s10).
scene(s2).	scene(s11).
scene(s3).	scene(s12).
scene(s4).	scene(s13).
scene(s5).	scene(s14).

Table 9: Positive and negative examples for learning *scene*

Background Knowledge		
on(s1,a,b).	rectangle(a).	square(b).
on(s2,f,e).	rectangle(f).	square(d).
on(s3,g,h).	rectangle(g).	square(h).
on(s4,q,r).	rectangle(q).	square(p).
on(s5,x,v).	rectangle(x).	square(v).
on(s10,l,k).	rectangle(k).	square(l).
on(s11,m,n).	rectangle(n).	square(o).
on(s12,u,r).	rectangle(r).	square(t).
on(s13,x1,x2).	rectangle(x1).	square(x3).
on(s14,x6,x5).	rectangle(x5).	square(x4).
triangle(c).	left_of(s1,b,c).	line(r).
triangle(e).	left_of(s2,d,e).	line(y).
triangle(i).	left_of(s3,h,i).	line(u).
triangle(j).	left_of(s4,p,r).	line(x2).
triangle(m).	left_of(s5,v,y).	line(x6).
	left_of(s10,j,k).	
	left_of(s11,n,o).	
	left_of(s12,r,t).	
	left_of(s13,x2,x3).	
	left_of(s14,x4,x5).	

Table 10: Background Knowledge for learning *scene*

With this sort of data *GOLEM* learns the following concept of *scene*

scene(A) :- on(A,B,C), left_of(A,D,E), rectangle(B), square(D).

It follows *GOLEM* activation (without tracing option) and output using files stem `scene`.

```
ianomami:/work1/ianomami/IA/carmo/golem/examples>>>../golem
!- doall(scene).
[Read in 83ms]
[REMOVED: 5 FORES, 0 NEGS]
[Foreground cover removed in 0ms]
[FORES: 0, BACKS: 55, NEGS: 2, RULES: 1]
[Clauses written to scene.r]
[Total time taken for induction = 249ms]
!- read(rules,scene).
!- show(rules).
```

```
scene(A) :- on(A,B,C), left_of(A,D,E), rectangle(B), square(D).
!-
```

One important aspect of any learning algorithm is related to the quality of the examples provided to learn the concept. Table 11 shows the various rules learned by *GOLEM* when different subsets of the set of negative examples {s10, s11, s12, s13, s14} are provided, showing that only negative examples {s13, s14} are necessary to learn the most specific expression of the concept.

Negative Examples					Learned Rules
S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	
					scene(A).
*					scene(A):-on(A,B,C),rectangle(B).
	*				scene(A):-on(A,B,C),rectangle(B).
		*			scene(A):-on(A,B,C),rectangle(B).
*	*	*			scene(A):-on(A,B,C),rectangle(B).
			*		scene(A):-left_of(A,B,C),square(B).
*			*		scene(A):-left_of(A,B,C),square(B).
*	*	*	*		scene(A):-left_of(A,B,C),square(B).
*			*	*	scene(A):-on(A,B,C),left_of(A,D,E),rectangle(B),square(D).
*	*	*	*	*	scene(A):-on(A,B,C),left_of(A,D,E),rectangle(B),square(D).
				*	scene(A):-on(A,B,C),rectangle(B).
			*	*	scene(A):-on(A,B,C),left_of(A,D,E),rectangle(B),square(D).

Table 11: Learned concept providing different negative examples

5.3 Learning the Concept *a_kind_of* — *ako*

This final example shows *GOLEM* capability to learn recursive predicates: in this particular case the predicate *a_kind_of*, which is a transitive relationship predicate. Figure 5 illustrates the example considered.

Tables 12 and 13 show respectively the positive and negative examples as well as background knowledge input to *GOLEM*.

Positive Examples	Negative Examples
ako(vertebrate,animais).	ako(simian,feline).
ako(fish,vertebrate).	
ako(mammal,vertebrate).	
ako(simian,primata).	
ako(carnivore,mammal).	
ako(tiger,mammal).	
ako(carnivore,animais).	
ako(carnivore,vertebrate).	
ako(feline,vertebrate).	
ako(herbivore,vertebrate).	
ako(mammal,animais).	
ako(feline,mammal).	

Table 12: Positive and negative examples for learning *ako*

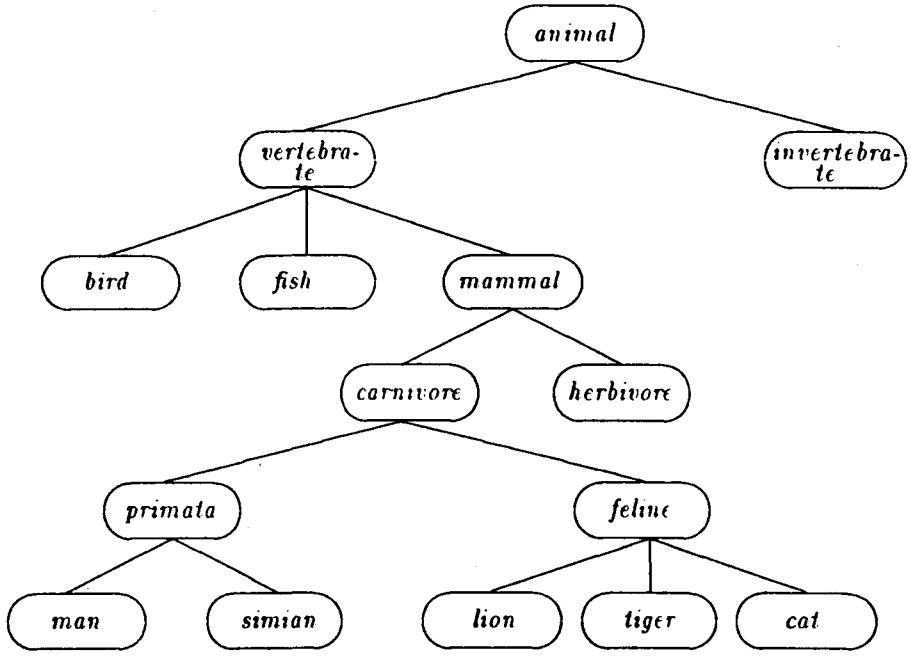

Figure 5: Animal hierarchy

Background Knowledge	
akol(invertebrate,animais)	ako(invertebrate,animais).
akol(vertebrate,animais).	ako(vertebrate,animais).
akol(bird,vertebrate).	ako(bird,vertebrate).
akol(fish,vertebrate).	ako(fish,vertebrate).
akol(mammal,vertebrate).	ako(mammal,vertebrate).
akol(carnivore,mammal).	ako(carnivore,mammal).
akol(herbivore,mammal).	ako(herbivore,mammal).
akol(primata,carnivore).	ako(primata,carnivore).
akol(feline,carnivore).	ako(feline,carnivore).
akol(man,primata).	ako(man,primata).
akol(simian,primata).	ako(simian,primata).
akol(lion,feline).	ako(lion,feline).
akol(tiger,feline).	ako(tiger,feline).
akol(cat,feline).	ako(cat,feline).
ako(tiger,mammal).	

Table 13: Background Knowledge for learning *ako*

The relation **akol/2** is a one pass relation, meaning that it relates each class to its immediately superior class. In the background knowledge, this relation is exhaustively provided. It should be observed as well that the description of the relation **ako/2** in the background knowledge is also exhaustively provided for the one pass description of this relation.

With this sort of data *GOLEM* learns the following concept of *ako*

```
ako(A,B) :- akol(A,B).
ako(A,B) :- akol(A,C), ako(C,B).
```

6 Conclusions

In this work we described in some details the ILP learning system *GOLEM*. While learning, this system is able to process large number of examples in a reasonable amount of time. It should be observed that despite its efficiency, *GOLEM* is highly dependent on the vocabulary and the form of examples given in advance, not being able to extend its vocabulary. Thus, as in any other inductive learning system, *GOLEM* heavily depends on the adequacy and quality of provided examples (negative and positive) as well as background knowledge in order to construct a suitable description of the learned concept.

GOLEM has an over-simplified way of handling noise; it allows a general clause C cover a user provided number of negative examples, independently of the number of positive examples covered by that clause. That is not a significative way of dealing with noise, since it does not take into consideration the relation between the number of positive and negative examples covered by the same clause C. It would be better if the user could provide the percentage of accuracy for constructing rules.

The implementation described in this work requires that the background knowledge K is given as ground facts. It would be interesting if the background knowledge could be given as a logical program P and through pre-processing, be transformed into a set of ground facts.

As shown in Section 2 this program P should be sintactically generative so to ensure that the model $\mathcal{M}$ for P is finite. We intent to implement a pre-processor which, for a given sintactically generative logical program P, constructs the corresponding *h-easy model* of P, for h provided by the user.

This approach is particularly interesting since a learned concept can be added to other background knowledge and eventually, through pre-processing, become part of a *h-easy model* used for learning another concept.

Acknowledgments: to Carmen Lucia Pagadigorria for the assistance in preparing the Figures contained in this Note.

References

[Aha 92] Aha D.W. *Relating Relational Learning Algorithms.* S. Muggleton (ed.) Inductive Logic Programming, Academic Press, 1992, pp 233-259.

[Buntine 88] Buntine W.L. *Generalized Subsumption and its Applications to Induction and Redundancy.* Artificial Intelligence, 36, 1988, pp 149-176.

[Cohen 93] Cohen, W.W. *Learnability of Restricted Logic Programs.* Proceedings of the Third International Workshop on Inductive Logic Programming — ILP'93, Bled, Slovenia, April 1993, pp 1-3.

[De Raedt 92] De Raedt, L. *Personal Communication.* 1992.

[Dolšak 92] Dolšak, B.; Muggleton, S. *The Application of Inductive Logic Programming to Finite-Element Mesh Design.* S. Muggleton (ed.) Inductive Logic Programming, Academic Press, 1992, pp 453-472.

[Feng 90] Feng, C.; Muggleton, S. *GOLEM 1.0 User's Commands.*, 1990.

[Frisch 90] Frisch, A.M.; Page, C.D. *Generalization with Taxonomic Information.* Proceedings of the Eighth National Conference on Artificial Intelligence, Boston, MA: AAAI Press, 1990, pp 755-761.

[Lavrač 92] Lavrač, N.; Džeroski, S. *Background Knowledge and Declarative Bias in Inductive Concept Learning.* Lectures Notes on Artificial Intelligence 462, 1992, pp 51-71.

[Lavrač 92] Lavrač, N.; Džeroski, S. *Inductive Learning of Relations from Noisy Examples.* S. Muggleton (ed.) Inductive Logic Programming, Academic Press, 1992, pp 495-516.

[Lavrač 94] Lavrač, N.; Džeroski, S. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

[Muggleton 90a] Muggleton, S.H. *Inductive Acquisition of Expert Knowledge*. Turing Institute Press, Glasgow, 1990.

[Muggleton 90] Muggleton, S.H.; Feng, C. *Efficient Induction of Logic Programs*. TIRM-90-044, Turing Institute Press, Glasgow, October 1990.

[Muggleton 91] Muggleton, S.H. *Inductive Logic Programming*. New Generation Computing, Vol. 8, 1991, pp 295-318.

[Nicoletti 93] Nicoletti, M.C.; Monard, M.C. *Herbrand Interpretation, Model and Least Model within the Framework of Logic Programming*. Notas ICMSC/USP Série Computação Nº 2, 30 pg., junho 1993.

[Page 92] Page, C.D.; Frish A.M. *Generalisation and Learnability: a Study in Constrained Atoms*. In S.H. Muggleton (ed.). Inductive Logic Programming, Academic Press, 1992. pp 29-61.

[Plotkin 71] Plotkin, G.D. *Automatic Methods of Inductive Inference*. Ph. D. thesis, Edinburgh University, August 1971.

[Quinlan-90] Quinlan, J.R. *Learning from Relational Data*. Proceedings of the 4th Austalian Joint Conference on Artificial Intelligence, World Scientific, November 1990, pp 38-47.

[Quinlan-91] Quinlan, J.R. *Knowledge Acquisition from Structured Data*. IEEE Expert, December 1991, pp 32-37.

[Utgoff 86] Utgoff, P.E. *Shift of Bias for Inductive Concept-learning*. R.S. Michalski; J.G. Carbonell and T.M. Mitchell (eds.). Machine Learning: An Artificial Intelligence Approach, Morgan Kaufmann, 1986.

[Wrobel 87] Wrobel, S. *Higher-order Concepts in a Tractable Knowledge Representation*. K. Morik, (ed.). GWAI-87 11th German Workshop on Artificial Intelligence, Informatik-Fachberichte Number 152, Springer-Verlag, October 1987, pp 129-138.

N O T A S D O I C M S C

Serie : Computacao

Nº 003/93 - ARENALES, M.N.; MORABITO, R.N. - An and/or-graph approach to the solution of two-dimensional non-guillotine cutting problems

Nº 002/93 - NICOLETTI, M.C.; MONARD, M.C. - Herbrand interpretation model and least within the framework of logic programming

Nº 001/93 - MORABITO, R.; ARENALES, M.N. - An and/or graph approach to the container loading problem