Predicate Invention and Learning from Positive Examples Only

Henrik Boström

Dept. of Computer and Systems Sciences
Stockholm University
Electrum 230, 164 40 Kista, Sweden
henke@dsv.su.se
Tel: +46-8-16 16 16
Fax: +46-8-703 90 25

Abstract. Previous bias shift approaches to predicate invention are not applicable to learning from positive examples only, if a complete hypothesis can be found in the given language, as negative examples are required to determine whether new predicates should be invented or not. One approach to this problem is presented, MERLIN 2.0, which is a successor of a system in which predicate invention is guided by sequences of input clauses in SLD-refutations of positive and negative examples w.r.t. an overly general theory. In contrast to its predecessor which searches for the minimal finite-state automaton that can generate all positive and no negative sequences, MERLIN 2.0 uses a technique for inducing Hidden Markov Models from positive sequences only. This enables the system to invent new predicates without being triggered by negative examples. Another advantage of using this induction technique is that it allows for incremental learning. Experimental results are presented comparing MERLIN 2.0 with the positive only learning framework of Progol 4.2 and comparing the original induction technique with a new version that produces deterministic Hidden Markov Models. The results show that predicate invention may indeed be both necessary and possible when learning from positive examples only as well as it can be beneficial to keep the induced model deterministic.

1 Introduction

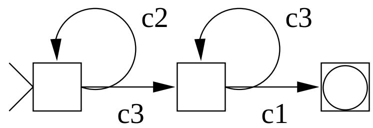
Bias shift approaches to predicate invention (e.g. [15, 6, 8, 1, 16]) introduce new predicates whenever the learning method fails to produce a consistent hypothesis in the given language [13]. This means that as long as it is possible to formulate a complete hypothesis in the given language (i.e. such that all positive examples are covered), negative examples are necessary for detecting inconsistency, and thus for inventing new predicates[1].

[1] Other approaches to predicate invention according to [13] fall outside the scope of this work. These are *reformulation approaches*, which uses predicate invention for optimising a theory w.r.t. size or compression, and *transformation approaches*, which base their decision on introducing new predicates on the operationality or efficiency of the induced hypothesis.

In this work we present one approach to this problem, the system MERLIN 2.0[2]. The system is a successor of MERLIN 1.0 [5], which uses an overly general theory to find SLD-refutations[3] of positive and negative examples, and then searches for the minimal finite-state automaton that can generate all sequences of input clauses in the SLD-refutations of the positive examples and none of the sequences in the SLD-refutations of the negative examples. For example, assume that we are given the following overly general theory T:

($c1$) `p([]).`
($c2$) `p([a|L]):- p(L).`
($c3$) `p([b|L]):- p(L).`

together with the positive examples $E^+ = \{$`p([a,a,a,b,b])`, `p([a,a,b,b,b,b])`$\}$ and the negative examples $E^- = \{$`p([a])`, `p([a,b,a])`$\}$. The corresponding positive sequences are $\{c2c2c2c3c3c1, c2c2c3c3c3c3c1\}$ and the negative sequences are $\{c2c1, c2c3c2c1\}$. The minimal automaton found by MERLIN 1.0 that is consistent with these sequences are:



This automaton is used together with the overly general hypothesis to construct the folllowing hypothesis:

```
p([a|A]) :- p(A).
p([b|A]) :- p_1(A).
p_1([]).
p_1([b|A]) :- p_1(A).
```

Note that without the invention of a new predicate (in this case `p_1`), it is not possible to formulate the hypothesis that the argument of p should be a list in which all a's are followed by one or more b's.

Clearly, if MERLIN 1.0 is given positive examples only, there is nothing that prevents the system from reducing the automaton to a one-state automaton that accepts all strings consisting of symbols that appear in the positive sequences. In the above example, this would result in a hypothesis identical to the overly general theory. In order to overcome this problem, MERLIN 2.0 uses a technique for inducing Hidden Markov Models from positive sequences only [14], that instead of minimising the size of the resulting automaton maximises the posterior probability. This enables MERLIN 2.0 to invent new predicates without being triggered by negative examples. In the next section, we present the technique for inducing Hidden Markov Models as well as a new extension which makes the induced Hidden Markov Model deterministic. In section three it is shown how

[2] The system can be obtained from `http://www.dsv.su.se/ML/MERLIN.html`
[3] Familiarity with logic programming terminology is assumed [9].

this technique is incorporated in MERLIN 2.0. In section four, we present experimental results comparing the system with positive only learning in Progol 4.2 [10, 11] as well as comparing the deterministic and non-deterministic versions of the technique for induction of Hidden Markov Models. Finally, in section five we give concluding remarks and point out directions for future research.

2 Induction of Hidden Markov Models

We first give a definition of Hidden Markov Models adopted from [14] and then briefly present the technique for inducing Hidden Markov Models that was introduced in [14].

2.1 Hidden Markov Models

Hidden Markov Models (HMMs) can be viewed as a stochastic generalisation of the non-deterministic finite automata (NFAs) [7]. As NFAs, HMMs accept (or generate) strings over the alphabet by non-deterministic walks between the initial and final states. In addition, HMMs also assign probabilities to the strings they generate, computed from the probabilities of individual transitions and emissions. These concepts are defined formally below.

Definition 1. A Hidden Markov Model is a quintuple $HMM = (Q, \Sigma, q_I, q_F, P)$ where Q is a set of states, Σ is an output alphabet, q_I is the initial state, q_F is the final state and P is a set of probability parameters, consisting of *transition probabilities* $p(q \rightarrow q')$ specifying the probability that state q' follows q, for all $q, q' \in Q$ and *emission probabilities* $p(q \uparrow \sigma)$ specifying the probability that symbol σ is emitted while in state q for all $q \in Q$ and $\sigma \in \Sigma$. It is assumed that $p(q \rightarrow q_I) = p(q_F \rightarrow q) = 0$ for all $q \in Q$ and $p(q_I \uparrow \sigma) = p(q_F \uparrow \sigma) = 0$ for all $\sigma \in \Sigma$.

Definition 2. An HMM is said to generate a string $x = \sigma_1\sigma_2 \cdots \sigma_l \in \Sigma^*$ if and only if there is a state sequence, or *path*, $q_I q_1 q_2 \cdots q_l q_F \in Q^*$ with non-zero probability, such that q_t outputs σ_t with non-zero probability for $t = 1, \ldots, l$. The *probability of a path* is the product of all transition and emission probabilities along it.

Definition 3. The *structure* or *topology* of an HMM consists of its states Q, its outputs Σ, a subset of its transitions $q \rightarrow q'$ with $p(q \rightarrow q') = 0$ and a subset of its emissions with $p(q \uparrow \sigma) = 0$.

Definition 4. The *conditional probability* $P(x|M)$ of a string $x = \sigma_1 \cdots \sigma_l$ given an HMM M is computed as the sum of the probabilities of all paths that generate x:

$$P(x|M) = \sum_{q_1 \ldots q_l \in Q^l} p(q_I \rightarrow q_1)p(q_1 \uparrow \sigma_1)p(q_1 \rightarrow q_2) \ldots p(q_l \uparrow \sigma_l)p(q_l \rightarrow q_F)$$

2.2 Induction of Hidden Markov Models

Traditional HMM estimation is based on the Baum-Welch algorithm [3], which assumes a certain topology and adjusts the parameters so as to maximise the model likelihood on the given samples. However, as we are primarily interested in finding the topology, and not the parameters, the technique in [14] is more appropriate than the former, as it in contrast to the former can be used for finding the HMM with maximimal posterior probability of the structure[4]. We first introduce the concept of posterior probability of an HMM structure according to [14] and then present their Best-first merging algorithm for finding an HMM with maximal posterior probability. Finally, we present an extension to the algorithm that forces the induced HMM to be deterministic.

Posterior probability for HMM structures We assume that there exists a distribution $P(M)$ independent of the data that assigns each model M an *a priori* probability, i.e. a bias. A model M can be decomposed into its structure part M_S and its parameter part θ_M, and the prior $P(M)$ can therefore be written as:

$$P(M) = P(M_S)P(\theta_M|M_S) \tag{1}$$

Given some data X, the problem is to find a model structure that maximises the posterior probability $P(M_S|X)$. Bayes' Law expresses the posterior as:

$$P(M_S|X) = \frac{P(M_S)P(X|M_S)}{P(X)} \tag{2}$$

Since the data X is fixed, this amounts to finding a model that maximises $P(M_S)P(X|M_S)$. Using the *Dirichlet distribution* as a parameter prior and a description length prior for the structure together with *the Viterbi approximation*[5] and the assumption that the Viterbi paths do not change as θ_M varies, the expression to be maximised, $P(M_S)P(X|M_S)$, can now be written:

$$\prod_{q \in Q} (|Q|+1)^{-n_t(q)}(|\Sigma|+1)^{-n_e(q)} F(t_{q_1}, \ldots, t_{q_{n_t(q)}}) F(e_{q_1}, \ldots, t_{q_{n_e(q)}}) \tag{3}$$

where t_{q_i} and e_{q_i} are the total counts of transitions and emissions, called *Viterbi counts*, occurring along the Viterbi paths associated with the samples in X and the n-dimensional function $F(t_1, \ldots, t_n)$ is defined as:

$$F(t_1, \ldots, t_n) = \frac{B(t_1 + 1/n, \ldots, t_n + 1/n)}{B(1/n, \ldots, 1/n)} \tag{4}$$

[4] The Baum-Welch algorithm can in principle also be used for finding a structure, as it may set some parameters to zero, but it requires that the maximal number of states is known and also that initial values are chosen for the model parameters, a choice of which the outcome of the algorithm is highly dependent.

[5] All paths except the most likely one, called the *Viterbi path*, are assumed to have zero probabibility.

where $B(\alpha_1, \ldots, \alpha_n)$ is the n-dimensional Beta function:

$$B(\alpha_1, \ldots, \alpha_n) = \frac{\Gamma(\alpha_1) \cdots \Gamma(\alpha_n)}{\Gamma(\alpha_1 + \cdots + \alpha_n)} \tag{5}$$

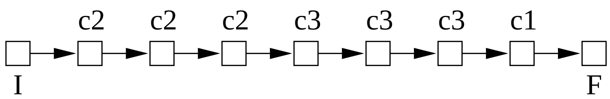
Best-first merging Below we present the incremental version of the Best-first merging algorithm in [14]. It takes as input a sequence of samples, incorporates them one by one into the current model and after each incorporation uses Hill-Climbing (with look-ahead) to find a new current model with maximal posterior probability by merging states. After all samples are processed, the current model is returned. Since the calculation of the posterior probability uses Viterbi counts, rather than transition and emission probabilities, such are kept by the algorithm.

The incorporation of a new sample $x = \sigma_1 \cdots \sigma_l$ into an existing model M results in that a set of new states $q_1, \ldots, q_l$ are added to Q and that the Viterbi counts for the transitions $q_I \rightarrow q_1$, $q_i \rightarrow q_i + 1$, $1 \leq i \leq l - 1$, and $q_l \rightarrow q_F$ and the emissions $q_i \uparrow \sigma_i$, $1 \leq i \leq l$, are set to one. When merging two states, the corresponding Viterbi counts are added and recorded as the counts for the new state.

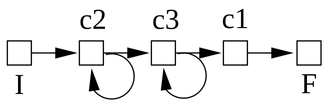
---

function Best-first merging($x_1 \cdots x_n, \mathit{LookAhead}$)
  M := the empty model
  for $i := 1$ **to** n **do**
    Incorporate x_i into M, $L := \mathit{LookAhead}$ and $B := M$
    repeat
      Let C be the set of models obtained from merging two states in B
      Let $B \in C$ be the model with maximal posterior probability $P(B|X)$
      if $P(B|X) > P(M|X)$ **then** $M := B$, $L := \mathit{LookAhead}$ **else** $L := L - 1$
    until $L = 0$ or $C = \emptyset$
  return M

Example After having incorporated one sample $x_1 = c2c2c2c3c3c3c1$ into the empty model, the resulting model is as follows:



One of the models with highest posterior probability obtained from merging two states is obtained by merging the last two states that emit `c3`. However, the posterior probability of this model is less than the initial model, but if look-ahead is allowed, the merging of the two states emitting `c3` in this model gives a model with higher posterior probability than the initial one. The merging process continues with the above model, eventually reaching the following model (using one-step look-ahead):[6]



Inducing Determistic HMMs Whenever a new sample is to be incorporated into the current model, a large part (or even all) of it may in fact already be accepted by the model, requiring little (or no) factual alterations to the model. The Best-first merging algorithm, however, generates a completely new sub-model for each sample and relies on the merging process to eventually incorporate the sample in the best way. In many cases, this is not only inefficient but may also mislead the Hill-Climbing search, as the chances of making the wrong choices increases with the number of states. The above approach can be viewed as being maximally pessimistic regarding the use of the current model for generating the new sample.

One could also consider a maximally optimistic version, which incorporates the sample into the existing model as far as possible. Assuming this means incorporating the longest prefix of the sample for which there is a path in the current model, this can be done efficiently by keeping the model *deterministic*, i.e. no state may have transitions leading to two different states that emit the same symbol. When incorporating a new sample, determinism is kept by aligning the sample as far as possible with the current model[7], introducing new states and transitions only for the suffix of the sample for which there is no path in the current model. When having merged two states, determinism can be kept by checking whether the new state has transitions leading to two different states that emit the same symbol, and if so, merging these two states. In section 4, we empirically compare the pessimistic with the optimistic approach.

3 MERLIN 2.0

Having induced an HMM that shows what sequences of input clauses are allowed in SLD-refutations of a given theory, MERLIN 2.0 produces a new theory that allows only those sequences that are allowed by both the original theory and the induced HMM. This is done by first converting the HMM into an NFA and by representing the set of sequences allowed by the given theory as a context-free grammar, and then deriving the intersection of the NFA and the first grammar. Finally, the intersection is used to produce the resulting hypothesis.

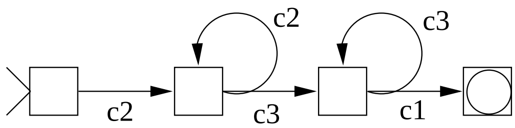
3.1 Converting an HMM into an NFA

The construction of an NFA from the structure of an HMM can be done in two ways: for each state, either the transitions leading from the state or the

[6] It should be noted that the term $P(X|M_S)$ prevents the HMM to be further reduced as the conditional probability of the sample would decrease.
[7] This includes updating the Viterbi counts.

transitions leading to the state are labeled with the symbols emitted by the state. We have chosen the latter option, since it avoids introducing transitions labeled with the empty string. Furthermore, all states with transitions leading to the final state in the HMM will become final states in the NFA.

Example The final HMM in the last example is converted into the following NFA:



3.2 Representing the Theory as a Context-Free Grammar

The set of possible sequences of input clauses in SLD-refutations of any instance of a goal G for a given program P can be represented by a context-free grammar, referred to as a *proof grammar*, (S, R), where S is the start symbol and R is a set of rules, where each rule is on the form $L \rightarrow CR_1 \ldots R_n$, where $n \geq 0$, $L, R_1 \ldots R_n$ are non-terminal symbols and C is a terminal symbol. Below, we show how to produce such a grammar by an example (for algorithmic details see [5]).

Example Given the goal `{:- p(L)}` together with the program in section 1, the above procedure produces the following proof grammar $(p/1, R)$, where R is the following set of rules:

$p/1 \rightarrow c1$
$p/1 \rightarrow c2\ p/1$
$p/1 \rightarrow c3\ p/1$

3.3 Deriving the Intersection

The intersection of a context-free language and a regular language is a context-free language [2]. In [5], a derivation of the algorithm in [2] is presented, which finds a context-free grammar that represents the intersection of a proof grammar and an NFA.

Example Given the NFA and the proof grammar in the previous examples, the following rules are produced by the procedure mentioned above, together with the start symbol $(p/1, q_0, \epsilon)$[8]:

[8] *Dead rules* have been removed.

$(p/1, q_0, \epsilon) \rightarrow c2\ (p/1, q_1, \epsilon)$
$(p/1, q_1, \epsilon) \rightarrow c2\ (p/1, q_1, \epsilon)$
$(p/1, q_1, \epsilon) \rightarrow c3\ (p/1, q_4, \epsilon)$
$(p/1, q_4, \epsilon) \rightarrow c1$
$(p/1, q_4, \epsilon) \rightarrow c3\ (p/1, q_4, \epsilon)$

3.4 Producing the Final Program

Having derived the intersection of the learned automaton and the original proof grammar, MERLIN 2.0 produces the final hypothesis in the form of a logic program, in which there is one clause for each rule in the intersection, and where each predicate symbol corresponds to a non-terminal symbol. This is achieved using the procedure presented in [5].

Example Given the context-free grammar in the previous example, together with the predicate symbol $p/1$, the above procedure produces the following program:

```
p([a|A]) :- p_1(A).
p_1([a|A]) :- p_1(A).
p_1([b|A]) :- p_2(A).
p_2([]).
p_2([b|A]) :- p_2(A).
```

4 Empirical Evaluation

In this section we present an empirical evaluation of the performance of MERLIN 2.0 both with the original Best-first merging algorithm and with the extension that forces the algorithm to produce deterministic HMMs[9]. MERLIN 2.0 is also compared to Progol 4.2, which is able to learn from positive examples only, but not to invent new predicates[10]. We first present the theories and example sets that were used in the experiments and then the experimental results.

[9] The lookahead was set to 5 for the non-deterministic version and to 1 for the deterministic version. Following [14], a logarithmic version of Bayes' law was used in the implementation including a global prior weight λ, giving $\lambda\, log\ P(M_S) + log\ P(X|M_S)$ as the quantity to be maximised. λ was set to 0.30 for the non-deterministic version and to 0.25 for the deterministic version. Furthermore, both versions first work in an incremental phase, in which states may only be merged if they have identical emissions and then in a second, non-incremental phase, in which all states may be merged.

[10] The default parameter settings were used in Progol except for the *posonly* parameter which was set to ON and the variable depth parameter (i) and the maximum clause length (c), which were set to 7.

4.1 Domains

The first theory that was investigated is the following:

($c1$) `nn(0).`
($c2$) `nn(s(X)):- nn(X).`

The entire set of examples consisted of the first 40 natural numbers, where the instances were classified as positive if they were odd, and negative otherwise (i.e. 50% positive examples). A correct definition of the odd numbers can be found without inventing new predicates, which means that Progol at least in theory is able to find it.

The second theory that was investigated is the same as presented in section 1, and the set of positive examples consisted of instances of `p(L)` where `L` contained up to 7 elements representing the regular expression $a^+b^+a^+$. A set of negative examples was generated by randomly replacing one of the elements in each positive example with the other constant, such that the new instance did not belong to the set of positive examples. The total number of examples in this set is 70 (of which 50% are positive). It should be noted that a correct hypothesis can not be produced for this domain without predicate invention.

The third theory extends the previous with one additional recursive clause, allowing the constant `c` to appear in the lists. The set of positive examples consisted of instances of `p(L)` where `L` contained up to 33 elements representing the regular expression $ac^*a \cup bc^*b$ (this target was also used in [14]). The set of negative examples were generated in the same way as for the previous domain, resulting in a set of 128 examples in total (of which 50% are positive).

The fourth theory and example set that were investigated were taken from [5], where the target predicate `turing(M)` represents a sequence of movements of a Turing machine, where each move is on the form $(Read, Write, Move)$ and the positive examples correspond to movements of a Turing machine performing addition. The total number of examples for this domain was 72 (of which 50% were positive).

4.2 Experimental Results

Each set of positive examples and each set of negative examples were randomly split into two halves. One half of the positive examples together with one half of the negative examples were used for testing. Subsets of the other half of positive examples were used for training, and the number of examples in these sets were varied, where a larger set always included a smaller. The same training and test sets were used for all three techniques. Each experiment was iterated 50 times and the mean accuracy on the test examples is presented below.

In Figure 1, the results from the odd number domain are presented. In this domain, the deterministic version of MERLIN 2.0 clearly outperforms the two other techniques, which produce overly general hypotheses (the mean number of clauses produced by Progol is 1.0 for all training sizes).

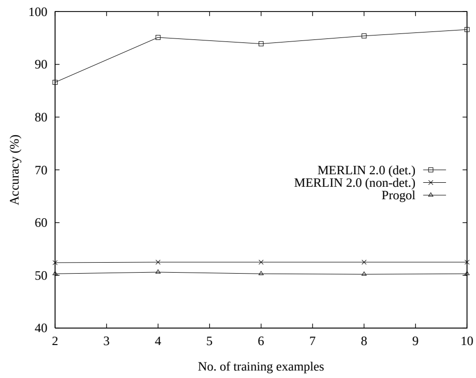


Fig. 1. Accuracy for the odd number domain.

In Figure 2, Figure 3 and Figure 4, the results from the $a^+b^+a^+$, $ac^*a \cup bc^*b$ and the Turing machine domains are presented. As for the previous domain, Progol produces overly general hypotheses in all three domains. That Progol would not perform well in these domains was expected as predicate invention is necessary for obtaining correct hypotheses. The deterministic version of MERLIN 2.0 outperforms the non-deterministic version in the first two of the three domains, due to that the non-deterministic version in the first case produces overly specific hypotheses and in the second case overly general hypotheses. In the last domain, the deterministic version suffers from producing overly specific hypotheses.

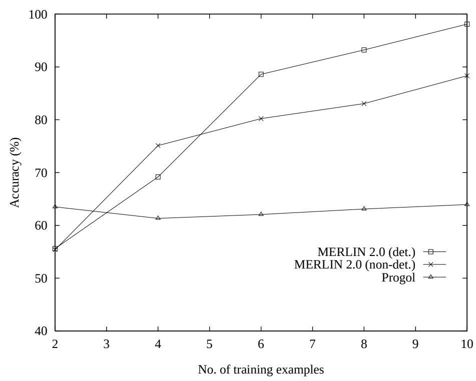


Fig. 2. Accuracy for the $a^+b^+a^+$ domain.

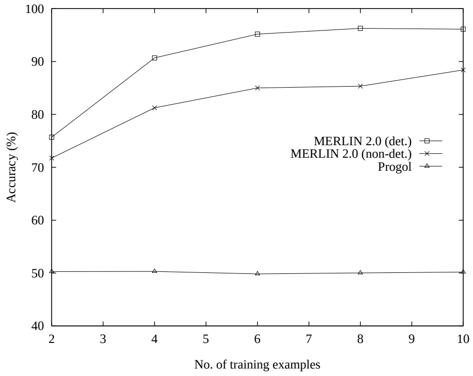


Fig. 3. Accuracy for the $ac^*a \cup bc^*b$ domain.

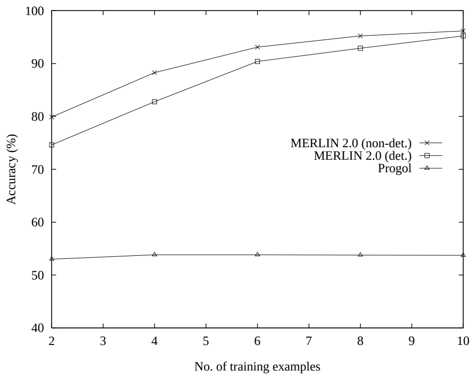


Fig. 4. Accuracy for the Turing machine domain.

5 Concluding Remarks

We have presented a novel approach to predicate invention when learning from positive examples only, the system MERLIN 2.0, which uses a technique for inducing Hidden Markov Models to determine when to invent new predicates. We have also proposed an extension to the induction technique, which makes the induced Hidden Markov Model deterministic. The system and the extension has been evaluated empirically, and the usefulness of predicate invention when learning from positive examples only was demonstrated as well as it was shown that it can be beneficial to keep the induced model deterministic.

There are a number of possible directions for future research. One is to experiment with the approach using other techniques for inducing finite-state automata from positive examples only (e.g. [4]). Another direction is to investigate extensions to the technique for inducing Hidden Markov Models, including techniques for finding a good global prior weight, using other search techniques than Hill-climbing and allowing negative examples. A third direction is to actually use the parameters which are set by the induction algorithm in order to induce stochastic logic programs [12].

Acknowledgements This work has been supported by the European Community ESPRIT Long Term Research Project no. 20237 *Inductive Logic Programming II* and the Swedish Research Council for Engineering Sciences (TFR).

References

1. Bain M. and Muggleton S., "Non-Monotonic Learning", in Muggleton S. (ed.), *Inductive Logic Programming*, Academic Press (1992) 145–161
2. Bar-Hillel Y., Perles M. and Shamir E., "On formal properties of simple phrase structure grammars", *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, **14**, 1, Akademie Verlag, Berlin (1961) 143–172
3. Baum L., Petrie T, Soules G. and Weiss N., "A maximization technique occurring in the statistical analysis of probabilistic functions in Markov chains", *The Annals of Mathematical Statistics* 41 (1970) 164–171
4. Biermann A. W. and Feldman J. A.," On the Synthesis of Finite-State Machines from Samples of Their Behavior", *IEEE Transactions on Computers* 21 (1972) 592–597
5. Boström H., "Theory-Guided Induction of Logic Programs by Inference of Regular Languages", *Proc. of the 13th International Conference on Machine Learning*, Morgan Kaufmann (1996) 46–53
6. Kijsirikul B., Numao M. and Shimura M., "Discrimination-based constructive induction of logic programs", *Proceedings of the 10th National Conference on Artificial Intelligence*, Morgan Kaufmann (1992) 44–49
7. Lewis H. R. and Papadimitriou C. H., *Elements of the Theory of Computation*, Prentice-Hall (1981)
8. Lapointe S., Ling, C. and Matwin S., "Constructive Inductive Logic Programming", *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann (1993) 1030–1036
9. Lloyd J. W., *Foundations of Logic Programming*, (2nd edition), Springer-Verlag (1987)
10. Muggleton S., "Inverse entailment and Progol", *New Generation Computing* **13** (1995) 245–286
11. Muggleton S., "Learning from positive data", *Proc. of the Sixth International Workshop on Inductive Logic Programming* (1996)
12. Muggleton S., "Stochastic Logic Programs", *Advances in Inductive Logic Programming* (Ed. L. De Raedt), IOS Press (1996) 254–264
13. Stahl I., "Predicate Invention in Inductive Logic Programming", *Advances in Inductive Logic Programming* (Ed. L. De Raedt), IOS Press (1996) 34–47
14. Stolcke A. and Omohundro S., "Best-first Model Merging for Hidden Markov Model Induction", TR-94-003, International Computer Science Institute, Berkeley, CA (1994)
15. Wirth R. and O'Rorke P., "Constraints on Predicate Invention", *Proceedings of the 8th International Workshop on Machine Learning* , Morgan Kaufmann (1991) 457–461
16. Wrobel S., "Concept Formation During Interactive Theory Revision", *Machine Learning Journal* **14** (1994) 169–192