A Structural GEM for Learning Logical Hidden Markov Models

Kristian Kersting[1], Tapani Raiko[2], and Luc De Raedt[1]

[1] Institute for Computer Science, Machine Learning Lab, University of Freiburg
Georges-Koehler-Allee 079, 79112 Freiburg, Germany
[2] Helsinki University of Technology, Laboratory of Computer and Information Science, P.O. Box 5400,02015 HUT, Finland

Abstract. Traditional hidden Markov models (HMMs) specify probability distributions over sequences of *flat* symbols. Recently, logical hidden Markov models (LOHMMs) have been introduced to deal with sequences of *structured* symbols. Within LOHMMs, logical atoms are used to represent output and state symbols. Together with unification, this kind of abstraction can lead to LOHMMs that have a much smaller number of parameters equivalent HMMs (typically an order of magnitude). However, the compactness of LOHMMs comes at the expense of a more complex structure learning problem. Indeed, different abstraction levels have to be explored. In this paper, we propose for the first time a method for learning the structure of LOHMMs from data. The method essentially adapts Friedman's *structural EM* (SEM) algorithm. More precisely, it combines *generalized expectation maximization* (GEM), which optimizes parameters, with structure search for model selection using *inductive logic programming* refinement operators. We provide convergence and experimental results that show its effectiveness.

Keywords: Multi-Relational Data Mining, Inductive Logic Programming, Expectation Maximization, Hidden Markov Models

1 Introduction

Hidden Markov models [26] (HMMs) are extremely popular for analyzing sequential data. Areas of application include computational biology, user modelling, speech recognition, empirical natural language processing, and robotics. Despite their successes, HMMs have a major weakness: they handle only sequences of flat, i.e., unstructured symbols. However, in many applications the symbols occurring in sequences are structured, e.g., in computational biology [17], web mining [1], information extraction from structured doucments, and user modelling. Consider, e.g., the sequence of UNIX commands `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... Such data has been used to train hidden Markov models for *anomaly detection* [19]. Anomaly detection is the task to develop a model or profile of the normal working state of a computer system user and to detect anomalous conditions as deviations from expected behaviour patterns. The user's current behavioral state, i.e., the hidden state, might be "idle", "writing", or "hacking". However, as the above command sequence shows, UNIX commands may have parameters (such as filenames), may return information (such as a return code) and may have other properties (such as current working directory, cost, etc.). Thus, commands are essentially *structured* symbols. Traditional HMMs cannot easily deal with this type of structured sequences. Typically, the application of HMMs requires either 1) ignoring the structure of the commands (i.e., the parameters), or 2) taking all possible parameters explicitly into account. The former approach results in serious information loss; the latter leads to a combinatorial explosion in the number of parameters and as a consequence inhibits generalization.

The above sketched problem with HMMs is akin to the problem of dealing with structured examples in traditional machine learning algorithms. This problem has extensively been studied in the fields of inductive logic programming [23] and multi-relational learning [6]. Recently, Kersting et. al [16] proposed *logical hidden Markov models* (LOHMMs) as an inductive logic programming approach to overcome the problem and have proven the usefulness of LOHMMs on a computational biology application [17]. The key idea underlying LOHMMs is to employ logical atoms as structured (output and state) symbols. Using logical atoms, the above UNIX command sequence can be represented as $\mathtt{emacs(lohmms.tex)}, \mathtt{ls}, \mathtt{latex(lohmms.tex)}, \ldots$ There are two important motivations for using logical atoms at the symbol level. First, *variables* in the atoms allow us to make abstraction of specific symbols. E.g., the logical atom $\mathtt{emacs(X,luc)}$ represents all files that user Luc could edit using emacs. Secondly, *unification* allows us to share information among hidden states and between hidden states and observations. E.g., the sequence $\mathtt{emacs(X,luc)}, \mathtt{latex(X,luc)}$ represents that the same file is used as an argument for both Emacs and LaTeX. Due to the use of abstraction, one can devise LOHMMs that are an order of magnitude smaller than equivalent HMMs. For instance, the trained LOHMM in [17] had only 120 parameters, corresponding to an instantiated traditional HMM with more than 62000 parameters.

However, the compactness of LOHMMs comes at the expense of a more complex structure learning, i.e., model selection problem. Indeed, different abstraction levels have to be explored. So far, no method for solving the problem has been proposed. This is a significant problem for several reasons. First, eliciting LOHMMs from experts can be a laborious and expensive process. Second, traditional hidden Markov models are commonly learned by estimating the maximum likelihood parameters of a fixed, fully connected model. Such an approach is not useful for LOHMMs because different levels of abstraction should be explored. Finally, logical hidden Markov models naturally contain *hidden* variables, i.e. no data case will describe the value of these variables. In such a case, learning is more difficult than in the *complete* data case. To evaluate the optimal choice of parameters for a candidate LOHMM, we must perform nonlinear optimization using e.g. the EM algorithm. In each iteration, the EM algorithm computes the probabilities of several events for each data case. Thus, learning parameters with the EM is significantly more difficult. The contribution of the present paper is a novel method for learning logical hidden Markov models. We tackle the problems by adapting Friedman's *structural EM* (SEM) algorithm [9, 10]. More precisely, our approach combines a generalized expectation maximization (GEM) algorithm, which optimizes parameters, with structure search for model selection using ILP refinement operators. In doing so, we explore different abstraction levels due to the *inductive logic programming* (ILP) refinement operators (see e.g. [23, 6]), and due to SEM's underlying idea, we reduce the learning problem to one that is similar to learning in the complete data case, which can be solved more efficiently.

The paper is organized as follows. In Section 2, we briefly review some logical concepts; in Section 3, we review logical hidden Markov models; in Section 4, we formalize the learning setting; in Section 5, we present a naïve learning algorithm; in Section 6, we introduce a structural, generalized EM for learning LOHMMs from data. The procedure is experimentally evaluated in Section 7. Before we conclude, we discuss related work.

2 Preliminaries

A *first-order logic alphabet* Σ is a set of relation symbols $\mathtt{r}$ with arity $m \geq 0$, written $\mathtt{r/m}$, and a set of functor symbols $\mathtt{f}$ with arity $n \geq 0$, written $\mathtt{f/n}$, a. If $n = 0$ then $\mathtt{f}$ is called a constant, if $m = 0$ then r is called a propositional variable. (We assume that at least one constant is given.) An *atom* $\mathtt{r}(\mathtt{t_1}, \ldots, \mathtt{t_m})$ is a relation symbol $\mathtt{r}$ followed by a bracketed m-tuple of terms $\mathtt{t_i}$. A *term* is a variable $\mathtt{V}$ or a functor symbol $\mathtt{f}$ of arity n immediately followed by a bracketed n-tuple of terms $\mathtt{s_j}$, i.e., $\mathtt{f}(\mathtt{s_1}, \ldots, \mathtt{s_n})$. An *iterative clause* is a formula of the form $\mathtt{A} \leftarrow \mathtt{B}$ where $\mathtt{A}$ and the $\mathtt{B}$ are logical atoms. In the present paper, a *logic program* consists of a set of iterative clauses. A substitution $\theta = \{\mathtt{V_1/t_1}, \ldots, \mathtt{V_k/t_k}\}$, e.g. $\{\mathtt{X/tex}\}$, is an assignment of terms $\mathtt{t_i}$ to variables $\mathtt{V_i}$. Applying a substitution σ to a term, atom or clause e yields the instantiated term, atom, or clause $e\sigma$ where all occurrences of the variables $\mathtt{V_i}$ are simultaneously replaced by the term $\mathtt{t_i}$, e.g. $\mathtt{ls(X)} \leftarrow \mathtt{emacs(F,X)}\{\mathtt{X}/tex\}$ yields $\mathtt{ls(tex)} \leftarrow \mathtt{emacs(F,tex)}$. A term, atom or clause e is called *ground* when it contains no variables, i.e., $vars(E) = \emptyset$. The *Herbrand base* of Σ, denoted as hb_Σ, is the set of all ground atoms constructed with the predicate and functor symbols in Σ. The set $G_\Sigma(\mathtt{A})$ of an atom $\mathtt{A}$ consists of all ground atoms $\mathtt{A}\theta$ that belong to hb_Σ.

3 Logical Hidden Markov Models

The logical component of a traditional hidden Markov model corresponds to a *Mealy machine* [13], i.e., to a finite state machine where the output symbols are associated with transitions [3]. This is essentially a propositional representation

[3] Often the output symbols are associated with states within HMMs. In this case, the logical component actually corresponds to a *Moore* machine. However, Moore and

because the symbols used to represent states and outputs are flat, i.e. not structured. The key idea underlying *logical* hidden Markov models is to replace these flat symbols by abstract symbols. An abstract symbol $\mathtt{A}$ is — by definition — a logical atom. It is abstract in that it represents the *set* of ground atoms $G_{\Sigma}(\mathtt{A})$. We assume that the alphabet is typed which means that constant symbols can appear as arguments of some particular relation symbols only. Therefore, assume a set $D_1, \ldots, D_n$ of (finite) domains (of constants), and let $\mathrm{D}(\mathtt{r/m})_i$ refer to the domain associated with the ith argument of a relation $\mathtt{r/m}$. Ground atoms then play the role of the traditional symbols used in a hidden Markov model.

Example 1 *Consider the alphabet Σ_1 which has as constant symbols* `tex`*,* `prog`*,* `hmm1`*, and* `lohmm1`*, and as relation symbols* `ls/0`*,* `emacs/1`*,* `latex/1`*,* `ls/1`*,* `emacs/2`*, and* `latex/2`*. Then the atom* $\mathtt{emacs(File,tex)}$ *represents the set* $\{\mathtt{emacs(hmm1,tex)}, \mathtt{emacs(lohmm1,tex)}\}$*. The alphabet is typed such that the constant symbols* `hmm1` *and* `lohmm1` *are used as filenames, i.e.,* $\mathrm{D}(\mathtt{latex/1})_1 = \mathrm{D}(\mathtt{latex/2})_1 = \mathrm{D}(\mathtt{emacs/1})_1 = \mathrm{D}(\mathtt{emacs/2})_1 = \{\mathtt{hmm1}, \mathtt{lohmm1}\}$ $= D_1$*, and* `tex` *and* `prog` *are used as users, i.e.,* $\mathrm{D}(\mathtt{ls/1})_2 = \mathrm{D}(\mathtt{latex/2})_2 = \mathrm{D}(\mathtt{emacs/2})_2 = \{\mathtt{tex}, \mathtt{prg}\} = D_2$*. We thus avoid useless instantiations such as* $\mathtt{emacs(tex,tex)}$*. Two different versions of the relation symbol encode that the* user *argument is not observed but only represented in the hidden states.*

The use of atoms instead of flat symbols allows us to analyze logical and structured sequences such as $\mathtt{emacs(hmm1)}, \mathtt{ls}, \mathtt{latex(hmm1)}$. Thus, the sequences of a logical hidden Markov model are structured. In addition, atoms are crucial in defining *abstract transitions.*

Definition 1 *(Abstract Transition) An* abstract transition *is an expression of the form* $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ *where* $p \in [0,1]$*, and* $\mathtt{H}$*,* $\mathtt{B}$ *and* $\mathtt{O}$ *are atoms.*

In this definition, the atoms $\mathtt{H}$ and $\mathtt{B}$ represent abstract states and $\mathtt{O}$ represents an abstract output symbol. The semantics of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ is that if one is in one of the states in $G_{\Sigma}(\mathtt{B})$, say $\mathtt{B}\theta_{\mathtt{B}}$, one will go to one of the states in $G_{\Sigma}(\mathtt{H}\theta_{\mathtt{B}})$, say $\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}}$, while emitting one of the output symbols in $G_{\Sigma}(\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}})$, say $\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}}$.

Example 2 *Consider the abstract transition* $0.8 : \mathtt{latex(File,tex)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File,tex)}$ *and assume that we are in state* $\mathtt{emacs(hmm1,tex)}$*, i.e.,* $\theta_{\mathtt{B}} = \{\mathtt{File/hmm1}\}$*. Then the abstract transition specifies that there is a probability of* 0.8 *that the next state will be in* $G_{\Sigma_1}(\mathtt{latex(hmm1,tex)}) = \{\mathtt{latex(hmm1,tex)}\}$ *(i.e., the probability is* 0.8 *that the next state will be* $\mathtt{latex(hmm1,tex)}$*), and that one of the output symbols in* $G_{\Sigma_1}(\mathtt{emacs(hmm1)}) = \{\mathtt{emacs(hmm1)}\}$ *(i.e.,* $\mathtt{emacs(hmm1)}$*) will be emitted.*

The above example was interesting because it uses unification between the filename arguments. On the other hand, it is simple because $\theta_{\mathtt{H}}$ and

Mealy machines are interconvertible. We decided to adapt Mealy machines because they fit our logical setting more intuitively.

$\theta_{\mathtt{O}}$ are both empty. The situation becomes more complicated when these substitutions are not empty because then the resulting state and output symbol sets are not necessarily singletons. Indeed, for the transition $0.8 : \mathtt{emacs(File', tex)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$ the resulting state set would be $G_{\Sigma_1}(\mathtt{emacs(File', tex)}) = \{\mathtt{emacs(hmm1, tex)}, \mathtt{emacs(lohmm1, tex)}\}$. Thus the transition is non-deterministic because there are two possible resulting states. We therefore need a mechanism to assign probabilities to these possible states. This is realized using the notion of a selection probability.

Definition 2 *(Selection Distribution) The* selection distribution μ *specifies for each abstract state (respectively observation)* $\mathtt{A}$ *over the alphabet* Σ *a distribution* $\mu(\cdot \mid \mathtt{A})$ *over the possible instantiations of* $\mathtt{A}$*, i.e., over* $G_\Sigma(\mathtt{A})$*.*

To continue our example, assume that we are in a state $\mathtt{latex(hmm1, tex)}$ and consider the transition $0.8 : \mathtt{emacs(File', tex)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$. Let $\mu(\mathtt{emacs(hmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.4$ and $\mu(\mathtt{emacs(lohmm1, tex)} \mid \mathtt{emacs(File', tex)}) = 0.6$. Then there would be a probability of $0.4 \times 0.8 = 0.32$ that the next state would be $\mathtt{emacs(hmm1, tex)}$ and of $0.6 \times 0.8 = 0.48$ that it would be $\mathtt{emacs(lohmm1, tex)}$.

Taking into account the selection distribution μ, the meaning of an abstract transition $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ can be summarized as follows. Let $\mathtt{B}\theta_{\mathtt{B}} \in G_\Sigma(\mathtt{B})$, $\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}} \in G_\Sigma(\mathtt{H}\theta_{\mathtt{B}})$ and $\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}} \in G_\Sigma(\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}})$. Then the model makes a transition from state $\mathtt{B}\theta_{\mathtt{B}}$ to $\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}}$ and emits symbol $\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}}$ with probability

$$p \cdot \mu(\mathtt{H}\theta_{\mathtt{B}}\theta_{\mathtt{H}} \mid \mathtt{H}\theta_{\mathtt{B}}) \cdot \mu(\mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}\theta_{\mathtt{O}} \mid \mathtt{O}\theta_{\mathtt{B}}\theta_{\mathtt{H}}). \tag{1}$$

To represent μ, any probabilistic representation can — in principle — be used, e.g. a Bayesian network. To reduce the model complexity, we will however use a naïve Bayes approach throughout the remainder of the this paper. In this approach, we associate to each domain D_i used as type a probability distribution P_{D_i} over D_i. Let $\mathrm{vars}(\mathtt{A}) = \{\mathtt{V_1}, \ldots, \mathtt{V_l}\}$ be the variables occurring in some atom $\mathtt{A}$ over $\mathtt{r/m}$, and let $\sigma = \{\mathtt{s_1/V_1}, \ldots \mathtt{s_l/V_l}\}$ be a substitution grounding $\mathtt{A}$. Each $\mathtt{V_j}$ is then considered a random variable over the domain of the first argument of $\mathtt{r/m}$ it appears in, denoted by $D_{\mathtt{V_j}}$. Then, $\mu(\mathtt{A}\sigma \mid \mathtt{A}) = \prod_{j=1}^{l} P_{D_{\mathtt{V_j}}}(\mathtt{V_j} = \mathtt{s_j})$. For instance for Σ_1, $\mu(\mathtt{emacs(hmm1, tex)} \mid \mathtt{emacs(F, E)})$ is computed as the product of $P_{D_1}(\mathtt{F} = \mathtt{hmm1})$ and $P_{D_2}(\mathtt{E} = \mathtt{tex})$.

So far, the semantics of a single abstract transition has been defined. A logical hidden Markov model will usually consist of multiple abstract transitions and this creates a further complication.

Example 3 *Consider the abstract transitions* $0.8 : \mathtt{latex(File, tex)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File, tex)}$ *and* $0.4 : \mathtt{ls(User)} \xleftarrow{\mathtt{emacs(File)}} \mathtt{emacs(File, User)}$*. These two transitions make conflicting statements about the state resulting from* $\mathtt{emacs(hmm1, tex)}$*. Indeed, according to the first transition, the probability is* 0.8 *that the resulting state is* $\mathtt{latex(hmm1, tex)}$ *and according to the second one it is* 0.4 *that it is* $\mathtt{ls(tex)}$*.*

This complication can be solved by taking into account the subsumption (or generality) relation among the B-parts of the two abstract transitions. Indeed, the B-part of the first transition $\mathtt{B}_1 = \mathtt{emacs(File, tex)}$ is more specific than that of the second transition $\mathtt{B}_2 = \mathtt{emacs(File, User)}$ because there exists a substitution $\theta = \{\mathtt{User/tex}\}$ such that $\mathtt{B}_2\theta = \mathtt{B}_1$, i.e., $\mathtt{B}_2$ subsumes $\mathtt{B}_1$. Therefore $G_\Sigma(\mathtt{B}_1) \subseteq G_\Sigma(\mathtt{B}_2)$. The first transition can therefore be regarded as more informative than the second one. It should therefore be preferred over the second one when starting from $\mathtt{emacs(hmm1, tex)}$. We will also say that the first transition is *more specific* than the second one. Remark that this *generality* relation imposes a partial order on the set of all transitions. These considerations lead to the strategy of only considering the maximally specific transitions that apply to a state in order to determine the successor states. This implements a kind of *exception handling* or *default reasoning*, e.g., in the above example, the first transition is an exception to the second one. This conflict resolution strategy will work properly provided that the bodies of all maximally specific transitions (matching the starting state) represent the same abstract state. This can be enforced by requiring the set of abstract transitions to be *well-founded* (for each predicate), i.e., by requiring that every subset of the set of abstract transitions has a unique maximally specific abstract body state. E.g., if the body of the second abstract transition in our example would have been replaced by $\mathtt{emacs(hmm1, User)}$ then the set of abstract transactions would not be well-founded.

By now we are able to formally define logical hidden Markov models.

Definition 3 *(Logical Hidden Markov Model) A* logical hidden Markov model *is a tuple $M = (\Sigma, \mu, \Delta)$ where Σ is a logical alphabet, μ a selection probability over Σ and Δ is a set of well-founded abstract transitions. Let $\mathbf{B}$ be the set of all atoms that occur as the body part of transitions in Δ. We require*

$$\forall \mathtt{B} \in \mathbf{B} : \sum_{\substack{\mathrm{cl} \in \Delta,\\ \mathrm{body(cl)} = \mathtt{B}}} \Pr(\mathrm{cl}) = 1.0. \tag{2}$$

The semantics of LOHMMs are as follows.

Theorem 1 (Semantics). *A logical hidden Markov model over a language Σ defines a discrete time stochastic process where the domain of the random variables is* $\mathrm{hb}(\Sigma)$*, i.e., the observation and hidden states. The induced probability measure over* $\mathrm{hb}(\Sigma)$ *is unique.*

Note that traditional HMMs are a special type of LOHMMs in which Σ contains only relation symbols of arity 0 and the selection probability is irrelevant. Thus, LOHMMs generalize traditional HMMs. However, LOHMMs allow to go beyond the expressivity of HMMs. Consider again the UNIX command sequence `emacs lohmms.tex, ls, latex lohmms.tex`. The filename `lohmms.tex` might be meaningless. What more matters is that both, emacs and LaTeX get the same file as input. The filename itself bears no information, i.e. it is an *identifier*. Assume a countably infinite set of constant symbols $i_1, i_2, \ldots$ Each time we would select a specific filename such as `lohmms.tex`, we select one of the i_j in increasing order if the filename `lohmms.tex` was not encountered before. Otherwise we

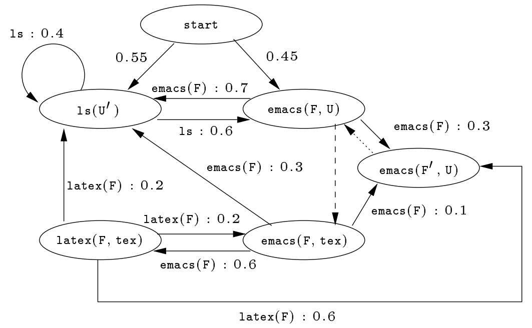

Fig. 1. A logical hidden Markov model.

select the same i_k, $k \leq j$, as before. What remains is the selection probability. Clearly, we do not want to specify a probability for each i_j. One solution is to use $P(i_1 \vee i_2 \vee \ldots)$. This translates to selecting a specific filename with out knowing which one. Consider e.g. our language Σ_1. Using identifiers instead of `hmm1` and `lohmm1`, the selection probability would be 1.0. Treating `lohmm1` exceptionally – e.g. because we would like to track it over time – its selection probability is p whereas that of `hmm1` would be $1-p$. In this way, a countably infinite set of (unspecific) inputs can be encoded in LOHMMs. Furthermore, using *functors* one can incorporate memory mechanisms into LOHMMs so that e.g. long distance correlations can be encoded.

Finally, let us note that logical hidden Markov models can be represented graphically. Figure 1 contains an example for analyzing Unix sequence commands. The underlying language Σ_2 consists of Σ_1 together with the constant symbols `other` denoting a not LaTeX user. In this graphical notation, abstract states are represented by vertices, abstract transitions by *solid edges*, and *dashed edges* represent the generality or subsumption ordering between abstract states. Furthermore, *dotted edges* connect identical abstract states. If we follow a transition to an abstract state with an outgoing dotted edge then the dotted edge will always be followed. Dotted edges are needed because the same abstract state can occur under different circumstances. Indeed, consider the transition $p : \mathtt{latex}(\mathtt{File}', \mathtt{User}') \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{User})$. Even though the atoms in the head and body of the transition are syntactically different they represent the same abstract state. Furthermore, to accurately represent the meaning of this transition we cannot use a solid edge from $\mathtt{latex}(\mathtt{File}, \mathtt{User})$ to itself, because this would actually represent the abstract transition $p : \mathtt{latex}(\mathtt{File}, \mathtt{User}) \xleftarrow{\mathtt{latex(File)}} \mathtt{latex}(\mathtt{File}, \mathtt{User})$, whose semantics is is different. Indeed, the second transition only allows for transitions between identical states whereas the first abstract transition also allows for transitions between equivalent states.

The graphical representation and the conflict resolution technique realize one of our design principles: locally interpretable transitions.

4 The Learning Setting

So far, we assumed that the LOHMM is given. Learning LOHMMs from data will often be easier than constructing them by hand. In this section, we investigate and formally define the problem of learning logical hidden Markov models.

Definition 4 *(Learning Problem)* **Given** *a set* $\mathbf{O} = \{O_1, \ldots, O_m\}$ *of data cases, a set* $\mathcal{M}$ *of LOHMMs, and a scoring function* $score_{\mathbf{O}} : \mathcal{M} \mapsto \mathbb{R}$, **find** *a hypothesis* $M^* \in \mathcal{M}$ *that maximizes* $score_{\mathbf{O}}$.

Each *data case* $O_i = \mathtt{o_{i,1}o_{i,2}\ldots o_{i,T}}$ consists of a sequence $\mathtt{o_{ij}}$ of observations, i.e. of ground atoms. We write each observation in lower case to stress that they are ground atoms. For instance in the user modelling domain an example data case is $\mathtt{emacs(lohmms), ls, emacs(lohmms)}$. By analogy with the traditional hidden Markov model learning setting, a single data case $O_i \in \mathbf{O}$ only describes the observations evolving over time. The corresponding sequence $H_i = \mathtt{h_{i,0}h_{i,1}\ldots h_{i,T_i+1}}$ describing the evolution of the system's state over time is hidden, i.e., not specified in O_i. For instance, we do not know whether $\mathtt{emacs(lohmms)}$ has been generated by $\mathtt{emacs(lohmms,prog)}$ or $\mathtt{emacs(lohmms,tex)}$ (cf. Example 3).

The *hypothesis space* $\mathcal{M}$ consists of all candidate LOHMMs to be considered during search. We assume Σ to be given. Consequently, the domain declarations, i.e. the possible constants which can be selected by μ are apriori known. Furthermore, each model $M \in \mathcal{M}$ can be considered to be parameterized by a vector $\boldsymbol{\lambda}_M$ such that each (legal) choice of values $\boldsymbol{\lambda}_M$ defines according to Theorem 1 a probability distribution $P(\cdot \mid M, \boldsymbol{\lambda}_M)$ over $\mathrm{hb}(\Sigma)$. For the sake of simplicity, we will denote the underlying logic program (i.e., the set of abstract transitions without associated probability values) by M and abbreviate $\boldsymbol{\lambda}_M$ by $\boldsymbol{\lambda}$ as long as the model M is clear from the context. Furthermore, we also impose certain syntactic restrictions, i.e., a syntactic bias on the transitions to be induced. At this point, we wish to stress that the particular syntactic bias selected is a parameter of our framework, see e.g. [24]

To evaluate different candidates M, we assume a scoring function $score_{\mathbf{O}} : \mathcal{M} \mapsto \mathbb{R}$ which evaluates how well a given candidate M fits a given set $\mathbf{O}$ of data cases. The data cases are assumed to be independently sampled from identical distributions. A commonly employed scoring function is

$$score_{\mathbf{O}}(M, \boldsymbol{\lambda}) = \log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) - Pen(M, \boldsymbol{\lambda}, \mathbf{O}). \tag{3}$$

The scoring function can be derived from Bayesian analysis assuming MAP-estimates, a uniform prior over the parameters and a prior over model structures that prefers simpler ones. The term $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda})$ is the *log-likelihood* of the data $\mathbf{O}$ given the current choice of model $(M, \boldsymbol{\lambda})$. The log-likelihood has a statistical interpretation: the higher the log-likelihood, the closer $(M, \boldsymbol{\lambda})$ models the probability distribution induced by the data. The second term $Pen(M, \boldsymbol{\lambda}, \mathbf{O})$ is a penalty function that biases the scoring function to prefer simpler models. Motivated by the *minimum description length* score for Bayesian networks [18], we use the simple penalty $Pen(M, \boldsymbol{\lambda}, \mathbf{O}) = \log(m)/2 \cdot |\Delta|$ in the present paper. The penalty is independent of the model parameters and therefore can neglect it during parameter estimation.

5 A Naïve Learning Algorithm

For traditional HMMs, the learning problem collapses to parameter estimation (i.e., estimating the transition probabilities) because HMMs are usually fully connected. For LOHMMs, however, we have to account for different abstraction levels. Therefore, we split the learning problem into

1. searching for a model structure, i.e., the underlying logic program, and
2. parameter estimating where the transition and selection probabilities consitute the parameters of a LOHMM,

and apply the following informed, greedy search algorithm. It is a direct solution to the learning problem described in Definition 4. It takes as input an initial model M^0 and the data $\mathbf{O}$, and performs:

1: Let $\boldsymbol{\lambda}^0 = \text{argmax}_{\boldsymbol{\lambda}}\ score_{\mathbf{O}}(M^0, \boldsymbol{\lambda})$
2: **Loop** for $k = 0, 1, 2, \ldots$
3: **Find** model $M^{k+1} \in \rho(M^k)\}$ that maximizes $score_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
4: Let $\boldsymbol{\lambda}^{k+1} = \text{argmax}_{\boldsymbol{\lambda}}\ score_{\mathbf{O}}(M^{k+1}, \boldsymbol{\lambda})$
5: **Until** convergence, i.e., no improvement in score

That is, at each stage k we choose a model structure and parameters among the currently best model and its neighbours (see below) that have the highest score. It stops, when there is no improvement in score. Note that in practice, we have to initialize the parameters of each model scored in lines 1 and 3 (e.g. randomly). To apply this algorithm to learning LOHMMs, we have to make line 3 more concrete. In the following two subsections, we will show (1) how to traverse the hypotheses space and (2) how to score hypotheses.

5.1 Traversing the Hypotheses Space

Let us start with the selection of the initial hypothesis M^0. It is governed by the idea that M^0 should in principle cover all possible observations and hidden state sequences (over the given language Σ). Therefore, an obvious candidate for M^0 (which we also used in our experiments) is the fully connected LOHMM built over all maximally general atoms over Σ. The maximally general atoms are expressions of the form $\mathtt{r}(\mathtt{X_1}, ..., \mathtt{X_m})$, where the $\mathtt{X_i}$ are different variables.

Now, to traverse the hypothesis space $\mathcal{M}$, we have to compute all neighbours of the currently best hypothesis M^k. To do so, we employ refinement operators traditionally used in inductive logic programming. More precisely, for the language bias considered and the experiments conducted in present paper, we used the refinement operator $\rho : \mathcal{M} \mapsto 2^{\mathcal{M}}$ which selects a single clause

$\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \mathcal{M}$ and adds a minimal specialization $\mathrm{cl}' \equiv p : \mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B}'$ of cl to $\mathcal{M}$ (w.r.t. to θ-subsumption). Specializing a single abstract transition amounts to instantiating and to unifying variables, i.e., $\mathrm{cl}' \equiv \mathrm{cl}\,\theta$ for some substitution θ. When adding cl' to M^k, we have to ensure that (1) all possible observations and hidden state sequences are covered and (2) the list of bodies $\mathbf{B}'$ after applying $\rho(M)$ should remain well-founded. Condition (1) can only be violated if $\mathtt{B}' \notin \mathbf{B}$. In this case, we add transitions with maximally general heads and observations. Condition (2) is established analogously. We complete the body lattice by adding new bodies (and therefore abstract transitions) in a similar way as described above. Both conditions together guarantee that the most specific body corresponding to a state is always unique.

5.2 Scoring Hypotheses

In principle, any maximum log-likelihood (ML) parameter estimation algorithm can be used to score hypotheses. However, in the presence of hidden variables ML estimation is a numerical optimization problem, and all known algorithms involve nonlinear, iterative optimization and multiple calls to an inference algorithm as subroutines. The most common ML parameter estimation technique for hidden Markov models is the Baum-Welch algorithm (see Appendix B for an adaptation) which is an instance of the Expectation-Maximization (EM) algorithm.

The EM algorithm [5] is a classical approach to maximum likelihood estimation in the presence of missing values. The basic observation underlying the EM algorithm is: we can easily find the ML parameters of a fixed model structure M^k if the data is complete, i.e., if we know the values for all the random variables. Therefore, it performs in each iteration $l+1$ two steps:

(E-Step) Based on the current parameters $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$, the algorithm computes a distribution over all possible completions of each partially observed data case.

(M-Step) Each completion is then treated as a fully observed data case weighted by its probability. New parameters $\boldsymbol{\lambda}^{k,l+1}$ are computed.

More formally, the E-step consists of computing the expectation of the log-likelihood given the old model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$, i.e.,

$$Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) = E\left[\log P(\mathbf{O}, \mathbf{H} \mid M^k, \boldsymbol{\lambda}) \mid M^k, \boldsymbol{\lambda}^{k,l}\right] . \tag{4}$$

Here, $\mathbf{O}, \mathbf{H}$ denotes the completion of the data cases $\mathbf{O}$. The current model $(M^k, \boldsymbol{\lambda}^{k,l})$ and the observed data $\mathbf{O}$ give us the conditional distribution governing the unobserved states $\mathbf{H}$. The expression $E[\cdot|\cdot]$ denotes the expectation over this conditional distribution. The function Q is called the *expected score*. In the M-step, the expected score $Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$ is maximized w.r.t. $\boldsymbol{\lambda}$, i.e.,

$$\boldsymbol{\lambda}^{k,l+1} = \mathrm{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda} | M^k, \boldsymbol{\lambda}^{k,l}) . \tag{5}$$

In fact, it is not necessary to maximize the expected score in each iteration. It is sufficient to choose $(M^k, \boldsymbol{\lambda}^{k,l+1})$ such that

$Q(M^k, \boldsymbol{\lambda}^{k,l+1} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$. Such an algorithm is called *generalized EM*. As Dempster et al. [5] have shown, the (generalized) EM algorithm improves the objective score in each iteration.

Theorem 2. *If* $Q(M^k, \boldsymbol{\lambda}^{k,l+1} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $score_{\mathbf{O}}(M^k, \boldsymbol{\lambda}^{k,l+1}) > score_{\mathbf{O}}(M^k, \boldsymbol{\lambda}^{k,l})$ *holds.*

Thus, if we choose in each iteration $(M^k, \boldsymbol{\lambda}^{k,l+1})$ that has a higher expected score than $(M^k, \boldsymbol{\lambda}^{k,l})$, we are bound to improve the objective score, see also [20]. The above naïve greedy algorithm can easily be instantiated to use the EM algorithm to estimate the ML parameters.

The problem with the naïve learning algorithm are its huge computational costs. In each structural iteration k, we evaluate all neighbours $M^{k'}$. For each neighbour, we need to run the EM algorithm for a reasonable number of iterations in order to get a reliable ML estimate of the parameters $\boldsymbol{\lambda}^{k'}$. Each EM iteration requires a full LOHMM inference on all given data cases. In total, the run time complexity per structural iteration is at least $\mathcal{O}(m \cdot \textit{EM iterations} \cdot \textit{costs of LOHMM inference})$ where *costs of LOHMM inference* depends on the length of data sequences, too. Friedman proposed the *structural EM* (SEM) to reduce the computational complexity [9, 10]. In the next section, we will show how to adapt the ideas underlying SEM to learning LOHMMs. We will follow the notation used in [9].

6 Structural Generalized EM

The idea underlying SEM is as follows. We take our current model $(M^k, \boldsymbol{\lambda}^k)$ and run the EM algorithm for a while to get reasonably completed data. We then fix the completed data cases and use them to compute the ML parameters $\boldsymbol{\lambda}^{k'}$ of each neighbour $M^{k'}$. We choose the neighbour with the best improvement of the score as $(M^{k+1}, \boldsymbol{\lambda}^{k+1})$ and iterate. More formally, we have

1: Initialize $\boldsymbol{\lambda}^{0,0}$ randomly
2: **Loop** for $k = 0, 1, 2, \ldots$
3:    **Loop** for $l = 0, 1, 2, \ldots$
4:       Let $\boldsymbol{\lambda}^{k,l+1} = \mathrm{argmax}_{\boldsymbol{\lambda}}\, Q(M^k, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
5:    **Until** convergence **or** $l = l_{\max}$
6:    **Find** model $M^{k+1} \in \rho(M^k)$ that maximizes $Q(\cdot \mid M^k, \boldsymbol{\lambda}^{k,l})$
7:    Let $\boldsymbol{\lambda}^{k+1,0} = \mathrm{argmax}_{\boldsymbol{\lambda}}\, Q(M^{k+1}, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})$
8: **Until** convergence

The hypotheses space is traversed as described in Section 5.1, and again we stop if there is no improvement in score. The following theorem shows that even when the structure changes in between, improving the expected score Q always improves the log-likelihood as well.

Theorem 3. *If* $Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l}) > Q(M^k, \boldsymbol{\lambda}^{k,l} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds, then* $\log P(\mathbf{O} \mid M, \boldsymbol{\lambda}) > \log P(\mathbf{O} \mid M^k, \boldsymbol{\lambda}^{k,l})$ *holds.*

The proof is a simple extension of the argumentation by [20, Section 3.2 ff.]. However, to apply this algorithm to learning LOHMMs, we still need to show how to choose the best neighbour, cf. line 6.

Let us first simplify the expected score in line 6. To do so, let $c(\mathtt{b},\mathtt{h},\mathtt{o})$ denote the counts, i.e., the number of times the systems proceeds from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. Now, it holds that

$$\begin{aligned}
Q(M,\boldsymbol{\lambda}|M^k,\boldsymbol{\lambda}^{k,l}) &= E\left[\log P(\mathbf{O},\mathbf{H}|M,\boldsymbol{\lambda})\Big|M^k,\boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\log \prod_t P(\mathtt{h_{t+1}},\mathtt{o_{t+1}}|\mathtt{h_t},M,\boldsymbol{\lambda})\Big|M^k,\boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\log \prod_{\mathtt{b},\mathtt{h},\mathtt{o}} P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})^{c(\mathtt{b},\mathtt{h},\mathtt{o})}\Big|M^k,\boldsymbol{\lambda}^{k,l}\right] \\
&= E\left[\sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} c(\mathtt{b},\mathtt{h},\mathtt{o})\cdot\log P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})\Big|M^k,\boldsymbol{\lambda}^{k,l}\right] \\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \underbrace{E\left[c(\mathtt{b},\mathtt{h},\mathtt{o})\Big|M^k,\boldsymbol{\lambda}^{k,l}\right]}_{=:ec(\mathtt{b},\mathtt{h},\mathtt{o})}\cdot\log P(\mathtt{h},\mathtt{o}|\mathtt{b},M,\boldsymbol{\lambda})\,. \qquad (6)
\end{aligned}$$

The term $ec(\mathtt{b},\mathtt{h},\mathtt{o})$ in (6) denotes the expected counts of making a transition from ground state $\mathtt{b}$ to ground state $\mathtt{h}$ emitting ground observation $\mathtt{o}$. The expectation is taken according to $(M^k,\boldsymbol{\lambda}^{k,l})$. Of course, it is sufficient to consider the expected counts of only those tuples $(\mathtt{b},\mathtt{h},\mathtt{o})$ which are possible given the data. All other tuples have some default expected counts, e.g. 0. The expected counts can be computed using the adapted Baum-Welch algorithm, see Appendix B.

In order to improve the parameters given the expected counts, we apply a gradient-based optimization technique to account for the subliminal nondeterminism LOHMMs possesses. Multiple abstract transitions with the same body can match a given tuple $(\mathtt{b},\mathtt{h},\mathtt{o})$ so that an analytical solution of the M-step seems to be difficult. In principle, a gradient-based optimization technique iteratively performs the following two steps. First, it computes the *gradient* vector $\nabla_{\boldsymbol{\lambda}}$ of partial derivatives of the scoring function w.r.t. the parameters of a LOHMM at a given point. Then, it takes a step in the direction of the gradient to the point $\boldsymbol{\lambda}+\delta\nabla_{\boldsymbol{\lambda}}$ where δ is the step-size. Thus, we have to compute the gradient vector. For LOHMMs, the gradient vector consists of partial derivatives w.r.t. abstract transition probabilities and to selection probabilities.

Assume that λ is the transition probability associated with some abstract transition cl. Now, the partial derivative of (6) w.r.t. some parameter λ is

$$\begin{aligned}
\frac{\partial Q(M,\boldsymbol{\lambda}\mid M^k,\boldsymbol{\lambda}^{k,l})}{\partial\lambda} &= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} ec(\mathtt{b},\mathtt{h},\mathtt{o})\cdot\frac{\partial\log P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}{\partial\lambda} \\
&= \sum\nolimits_{\mathtt{b},\mathtt{h},\mathtt{o}} \frac{ec(\mathtt{b},\mathtt{h},\mathtt{o})}{P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}\cdot\frac{\partial P(\mathtt{h},\mathtt{o}\mid\mathtt{b},M,\boldsymbol{\lambda})}{\partial\lambda} \qquad (7)
\end{aligned}$$

The partial derivative of $P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})$ w.r.t. λ can be computed as follows [4]:

$$
\begin{aligned}
&\frac{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})}{\partial \lambda} = \\
&= \frac{\partial}{\partial \lambda} \sum\nolimits_{\mathrm{cl}} P(\mathrm{cl} \mid M, \boldsymbol{\lambda}) \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \\
&= \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \qquad (8)
\end{aligned}
$$

Substituting (8) back into (7) yields

$$
\begin{aligned}
&\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda} \\
&= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \left(\frac{ec(\mathtt{b}, \mathtt{h}, \mathtt{o})}{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \right) \qquad (9)
\end{aligned}
$$

The selection probability follows a naïve Bayes approach. Therefore, one can show in a similar way as for transition probabilities that

$$
\begin{aligned}
&\frac{\partial Q(M, \boldsymbol{\lambda} \mid M^k, \boldsymbol{\lambda}^{k,l})}{\partial \lambda} \\
&= \sum_{\mathtt{b},\mathtt{h},\mathtt{o}} \Big(\frac{ec(\mathtt{b}, \mathtt{h}, \mathtt{o})}{P(\mathtt{h}, \mathtt{o} \mid \mathtt{b}, M, \boldsymbol{\lambda})} \cdot \sum_{\mathrm{cl}} c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o}) \cdot P(\mathrm{cl} \mid M, \boldsymbol{\lambda}) \cdot \qquad (10) \\
&\qquad \mu(\mathtt{h} \mid \mathrm{head}(\mathrm{cl})\sigma_{\mathtt{h}}, M) \cdot \mu(\mathtt{o} \mid \mathrm{obs}(\mathrm{cl})\sigma_{\mathtt{h}}\sigma_{\mathtt{o}}, M) \Big) \qquad (11)
\end{aligned}
$$

where $c(\lambda, \mathrm{cl}, \mathtt{b}, \mathtt{h}, \mathtt{o})$ is the number of times that the domain element associated with λ is selected to ground cl w.r.t. $\mathtt{h}$ and $\mathtt{o}$.

Note that in the problem at hand, the described method has to be modified to take into account that the parameter vector $\boldsymbol{\lambda}$ consists of probability values, i.e. $\lambda \in [0, 1]$ and that corresponding parameters sum to 1.0. There are two ways to enforce this: (1) projecting the gradient onto the constraint surface, and (2) reparameterizing the problem so that the new parameters automatically respect the constraints on $\boldsymbol{\lambda}$ no matter what their values are. We choose the latter approach as the reparameterized problem is fully unconstrained. More precisely, we define the parameters $\beta_{ij} \in \mathbb{R}$ for a set of corresponding λ's such that $\lambda_{ij} = (\beta_{ij}^2)/(\sum_k \beta_{ik}^2)$. This enforces the constraints given above, and a local maximum w.r.t. $\boldsymbol{\beta}$ is also a local maximum w.r.t. $\boldsymbol{\lambda}$, and vice versa. The gradient w.r.t the β_{ij}'s can be found by computing the gradient w.r.t the λ_{ij}'s and then deriving the gradient w.r.t. $\boldsymbol{\beta}$ using the chain rule, see e.g. [2, 15].

What do we gain from the structural GEM over the original naïve learning algorithm? We save many runs of the EM algorithm because we use the same ground counts for scoring all neighbouring hypotheses. Each iteration of the

[4] For the sake of simplicity, we will not check in the following equations that a transition is *maximally specific* for some ground states as done in the algorithms presented in the appendix.

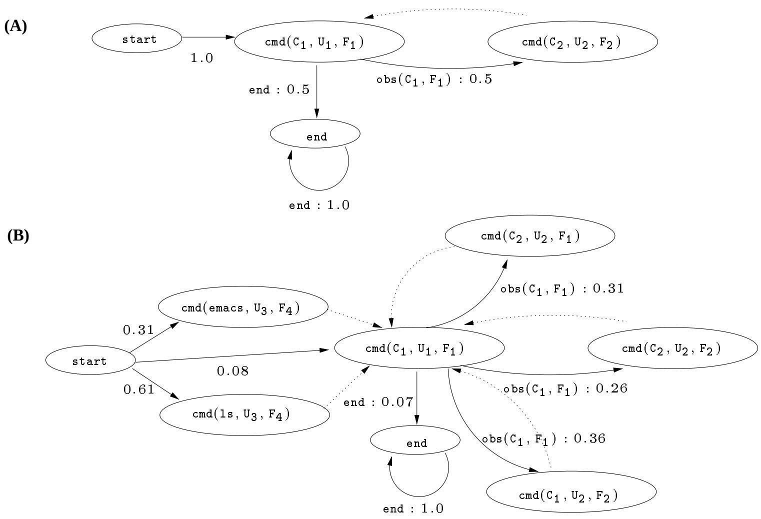

Fig. 2. The initial logical hidden Markov model (A) and the learned one (B).

gradient algorithm is computationally less expensive than one EM iteration. Essentially, considering the evaluation of the neighbours, we have made the run time complexity independent of the number and length of the data cases — a feature which is important for scaling up. However, maximizing the expected score (6) based on a different model structure does not only maximize likelihood but also tries to minimize the difference in the distribution of the hidden states sequences between the old and the new hypothesis. The SEM and SGEM learning schemes thus prefer changes in the model structure that do not change the role of the hidden states to those that do. Nevertheless, we believe that the expected score is a good heuristic in learning.

7 Experimental Prospects

The proposed method is intended as a generic framework for learning logical hidden Markov models. As such, it leaves several issues open. These include: the actual language bias (and corresponding refinement operator), the scoring function, possible pruning methods, etc. Nevertheless, in order to show the validity of our framework, we report on some experiments that proves that the underlying principles work. To this aim, we made a simple implementation in Sicstus Prolog 3.9.0. The implementation features a beam-search using the simple penalized log-likelihood (see Section 4) of the data as score. The chosen syntactic language bias did not allow for functors to be used. To perform the maximum likelihood estimation, we implemented the Baum-Welch algorithm as described in Appendix B (but scaled to avoid numerical imprecision). For the improvement of expected score, we adapted the scaled conjugate gradient (SCG) as implemented in Bishop and Nabney's Netlab library (`http://www.ncrg.aston.ac.uk/netlab/`,

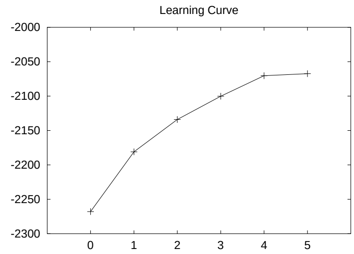

Fig. 3. Iteration (x axis) versus the penalized log-likelihood (y axis) achieved.

see also [3]). SCGs are well-known from the field of learning neural networks and avoid the time consuming line search of more traditional gradient-based methods by employing an approximation of the Hessian of the scoring function to quadratically extrapolate the minimum instead of doing a line search. The number of expected score evaluations is at most twice per iteration. We set $l_{\max} = 10$. Within the gradient-based optimization, we stopped when a limit of 10 iterations was exceeded or a change in penalized log-likelihood (reps. expected score) was less than 10^{-2} from one iteration to the next.

Data were generated from a LOHMM inspired by the user-modelling LOHMM, cf. Figure 1. Instead of modelling each command by its own predicate, we used the predicates `cmd(Command,User,Filename)` (to model internal states) and `obs(Command,Filename)` (to model the observations). There were 2 filenames, 2 users types, and 3 commands. From this model, we sampled 100 sequences of length 20. We selected the LOHMM shown in Figure 2 (A) as initial hypothesis. The learned model is shown in Figure 2. The learning curve in Figure 3 clearly proves the principle: the objective score increases in each iteration. The learned model is interesting for three reasons. First, it has less parameters than a a fully connected HMM modelling this domain. The latter consists of 156 parameters whereas the learned LOHMM consists of 20 parameters only. Second, the algorithm managed to adjust the starting probability distribution (`start` node). Third and more importantly, it unifies variables to model the distribution more accurately. For example, sequences of commands (even more than two) using the same filename are ran. Such knowledge cannot easily be read off from traditional hidden Markov models.

8 Related Work

LOHMMs can be analyzed from two different directions. On the one hand, they are related to several extensions of HMMs that have been investigated, such as hierarchical HMMs [7], factorial HMMs [12], and HMMs based on tree automata [8]. On the other hand, they are also related to the recent interest in combining inductive logic programming [23] principles with probability theory, see [4] for an overview.

In the first type of approach, the underlying idea is to decompose the state variables into smaller units, i.e. to *upgrade* HMMs to represent more structured state spaces. In hierarchical HMMs states themselves can be HMMs, in factorial HMMs they can be factored into k state variables which depend on one another only through the observation[5], and in tree based HMMs the represented probability distributions are defined over tree structures. The key difference with LOHMMs is that these approaches do not employ the logical concept of unification. Unification is essential because it allows us to introduce abstract transitions, which do not *consist* of more detailed states. Thus, structure learning algorithm devised for these approaches cannot directly applied to LOHMMs.

In the second type of approach, most attention has been devoted to developing highly expressive formalisms, such as e.g. stochastic logic programs [22], PRISM [27], probabilisitic relational models [11], and Bayesian logic programs [14]. LOHMMs can be seen as an attempt towards *downgrading* such highly expressive frameworks. As a consequence, LOHMMs represent an interesting position on the expressiveness scale. Whereas they retain most of the essential logical features of the more expressive formalisms, they seem easier to understand, adapt and learn. This is akin to many contemporary considerations in *inductive logic programming* [23] and multi-relational data mining [6]. In any case, to the best of our knowledge no structure learning algorithm based on the *structural EM* as been proposed.

LOHMMs are most closely related to the recently introduced *relational Markov models (RMMs)* [1]. Here, states can be of different types, with each type described by a different set of variables. The domain of each variable can be hierarchically structured. The main differences are that RMMs do not facilitate variable binding and unification nor hidden states. Therefore, they are more a first type than a second type approach. Furthermore, they do not select the most-specific transition to resolve conflicting transitions. Instead, they interpolate between conflicting ones. This is an interesting option for LOHMMs because it makes parameter estimation more robust. On the other hand, it also seems to make it more difficult to adhere one of our design princples: locally interpretable transitions. Structure learning has been addressed for RMMs based on *probability estimation trees* [25], a kind of decision trees.

Finally, our learning approach is to some extend contrary to some more advanced technique for learning the structure of hidden Markov models, namely model merging, see e.g. [28]. Here, the induction process starts with the most specific model consistent with the training data and generalizes by successively merging states. However, abstract transitions aim at good generalization, and the most general clauses can be considered to be the most informative ones.

[5] Factorial HMMs could be viewed as a special type of LOHMMs, where the hidden state is summarized by a $2 \cdot k$-ary abstract state. The first k arguments encode the k state variables, and the last k arguments serve as a memory of the previous joint state. The selection distribution of the i-th argument is conditioned on the $i + k$-th argument.

9 Conclusions

A method for learning logical hidden Markov models has been presented. The method essentially adapts Friedman's *structural EM* algorithm. More precisely, it combines standard *generalized EM*, which optimizes parameters, with structure search for model selection using refinement operators from the field of inductive logic programming. Experiments have been presented that show that the presented techniques work in principle.

As future work, we plan to conduct a more thorough experimental evaluation. The experiments were realized by relying on a simple and basic instantiation of our framework. Future work will be concerned with making more refined and state-of-the-art choices for many parameters. These include: the investigation of other objective scores, other refinement operators e.g. handling functors, deleting transitions, and generalizing hypotheses, logical pruning criteria for hypotheses, the incorporation of syntactic bias mechanisms, efficient storing of ground counts (e.g. using AD-trees [21]) etc. Nevertheless, the authors hope that the presented framework will inspire further research at the intersection of inductive logic programming and hidden Markov models.

References

1. C. R. Anderson, P. Domingos, and D. S. Weld. Relational Markov Models and their Application to Adaptive Web Navigation. In D. Hand, D. Keim, O. R. Zaïne, and R. Goebel, editors, *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 143–152, Edmonton, Canada, 2002. ACM Press.
2. J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive Probabilistic Networks with Hidden Variables. *Machine Learning*, 29(2–3):213–244, 1997.
3. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford Univ. Press, 1995.
4. L. De Raedt and K. Kersting. Probabilistic Logic Learning. *SIGKDD Explorations: Special Issue on Multi-Relational Data Mining*, 5(1), 2003. (To appear).
5. A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B 39:1–39, 1977.
6. S. Džeroski and N. Lavrač. *Relational Data Mining*. Springer-Verlag, 2001.
7. S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden Markov model: analysis and applications. *Machine Learning*, 32:41–62, 1998.
8. P. Frasconi, G. Soda, and A. Vullo. Hidden Markov Models for Text Categorization in Multi-Page Documents. *Jour. of Intel. Information Systems*, 18:195–217, 2002.
9. N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In D. H. Fisher, editor, *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-1997), Nashville, Tennessee, USA, July 8-12, 1997*, pages 125–133. Morgan Kaufmann, 1997.
10. N. Friedman. The Bayesian structural EM algorithm. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 129–138, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

11. N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI-99)*, pages 1300–1309, Stockholm, Sweden, 1999. Morgan Kaufmann.
12. Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.
13. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
14. K. Kersting and L. De Raedt. Bayesian Logic Programs. Technical Report 151, University of Freiburg, Institute for Computer Science, April 2001. (submitted).
15. K. Kersting and N. Landwehr. Scaled Conjugate Gradients for Maximum likelihood: An Empirical Comparison with the EM Algorithm. In J. A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02)*, pages 89–98, Cuenca, Spain, 2002. `http://www.info-ab.uclm.es/isl/cuenca/`.
16. K. Kersting, T. Raiko, and L. De Raedt. Logical Hidden Markov Models (Extended Abstract). In *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02)*, pages pp. 99–107, Cuenca, Spain, November 6–8 2002.
17. K. Kersting, T. Raiko, S. Kramer, and L. De Raedt. Towards discovering structural signatures of protein folds based on logical hidden Markov models. In R. B. Altman, A. K. Dunker, L. Hunter, T. A. Jung, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 192 – 203, Kauai, Hawaii, USA, 2003. World Scientific.
18. W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Computational Intelligence*, 10(4), 1994.
19. T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. In Åsa Rudström, editor, *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pages 35–44, Stockholm, Sweden, July 1999.
20. G. J. McKachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Eiley & Sons, Inc., 1997.
21. Andrew Moore and Mary Soon Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, March 1998.
22. S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.
23. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.
24. C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. Declarative Bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*. IOS Press, 1996.
25. F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.
26. L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.
27. T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, 15:391–454, 2001.
28. A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 11–18. Morgan Kaufman, 1993.

A Evaluation of LOHMMs

Consider the probability of the partial observation sequence $O = \mathtt{o_1 o_2 \ldots o_T}$ and (ground) state $\mathtt{h}$ at time t, given the model M, i.e. $\alpha_t(\mathtt{h}) := P(\mathtt{o_1 o_2 \ldots o_t}, q_t = \mathtt{h} \mid M)$ where $q_t = \mathtt{h}$ denotes that the system is in state $\mathtt{h}$ at time t. Clearly, $P(O \mid M) = \sum_{\mathtt{h} \in S_{T+1}} \alpha_{T+1}(\mathtt{h})$ where S_t denotes the set of reachable states at time t. As for traditional HMMs, $\alpha_t(\mathtt{h})$ can be computed using a dynamic programming approach. The only difference is that in LOHMMs the substitutions have to be taken into account.

$S_0 := \{\mathtt{start}\}$ /* *initialize the set of reachable states* */
$\alpha_0(\mathtt{start}) := 1.0$ /* *initialize the α value for start* */
for $t = 1, 2, \ldots, T+1$ **do**
  $S_t = \emptyset$ /* *initialize the set of reachable states at clock t* */
  foreach $\mathtt{b} \in S_{t-1}$ **do**
    foreach maximally specific $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ such that $\sigma_{\mathtt{b}} = mgu(\mathtt{b}, \mathtt{B})$ exists **do**
      foreach $\mathtt{h} = \mathtt{H}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}} \in G_\Sigma(\mathtt{H}\sigma_{\mathtt{b}})$ such that $\mathtt{o_{t-1}}$ unifies with $\mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}}$ **do**
        if $\mathtt{h} \notin S_t$ **then**
          $S_t := S_t \cup \{\mathtt{h}\}$
          $\alpha_t(\mathtt{h}) := 0.0$
        $\alpha_t(\mathtt{h}) := \alpha_t(\mathtt{h}) + \alpha_{t-1}(\mathtt{b}) \cdot p \cdot \boxed{\mu(\mathtt{h} \mid \mathtt{H}\sigma_{\mathtt{b}}) \cdot \mu(\mathtt{o_{t-1}} \mid \mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}})}$
return $P(O \mid M) = \sum_{\mathtt{s} \in S_{T+1}} \alpha_{T+1}(\mathtt{s})$

The boxed terms constitute the main difference to the corresponding HMM formula. The computational complexity is $\mathcal{O}((T+1) \cdot s \cdot (|\Delta| + o \cdot g))$ where $s = \max_{t=1,2,\ldots,T+1} |S_t|$, $|\Delta|$ is the number of abstract transitions, o is the maximal number of outgoing abstract transitions with regard to an abstract state, and g is the maximal number of ground instances of an abstract state. In a completely analogous manner, one can devise a *backward procedure* to compute $\beta_t(\mathtt{h}) = P(\mathtt{o_{t+1} o_{t+2} \ldots o_T} \mid q_t = \mathtt{h}, M)$. Based on $\alpha_t(\mathtt{h})$ and $\beta_t(\mathtt{h})$, we can devise an adaption of the Baum-Welch algorithm.

B The Baum-Welch Algorithm for LOHMMs

We have to estimate the maximum likelihood transition probabilities and selection distributions. To estimate the former ones, we upgrade the *Baum-Welsh* approach for HMMs. There, an improved estimate $\overline{p}$ of the transition probability of some (ground) transition $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ is computed by taking the ratio between the expected number $\xi(\mathrm{cl})$ of making the transitions cl at any time given the model M and an observation sequence O, and the total number of transitions starting from $\mathtt{B}$ at any time given the model M and an observation sequence O. Basically the same applies when cl is an abstract transition, however we have to be a little bit more careful because we have no direct access to $\xi(\mathrm{cl})$. To compute

$\xi_t(\mathrm{cl})$ in case that cl is abstract, let $\xi_t(\mathrm{gcl}, \mathrm{cl})$ be the probability of following the abstract transition cl via its ground instance $\mathrm{gcl} \equiv p : \mathtt{h} \xleftarrow{\mathtt{o}} \mathtt{b}$, i.e.

$$\xi_t(\mathrm{gcl}, \mathrm{cl}) = \frac{\alpha_t(\mathtt{b}) \cdot p \cdot \beta_{t+1}(\mathtt{h})}{P(O \mid M)} \cdot \boxed{\mu(\mathtt{h} \mid \mathtt{H}\sigma_{\mathtt{b}}) \cdot \mu(\mathtt{o_{t-1}} \mid \mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}})} \tag{12}$$

where $\sigma_{\mathtt{b}}$, $\sigma_{\mathtt{h}}$, and $\sigma_{\mathtt{o}}$ are defined as before, and $P(O \mid M)$ is the probability that the model generated the sequence O. Again, the boxed terms constitute the main difference to the corresponding HMM formula. Now, it holds that $\xi_t(\mathrm{cl}) = \sum_{\mathrm{gcl}} \xi_t(\mathrm{gcl}, \mathrm{cl})$ where the sum runs over all ground instances of cl. This leads to the following reestimation formula where we assume that the sets S_t of reachable states are reused from the computations of the α- and β-values:

1: */* initialization of expected counts */*
2: **foreach** $\mathrm{cl} \in \Delta$ **do**
3: $\quad \xi(\mathrm{cl}) := 1$ */* or 0 if not using simple pseudocounts */*
4: */* compute expected counts */*
5: **foreach** $t = 0, 1, \ldots, T$ **do**
4: $\quad$ **foreach** $\mathtt{b} \in S_t$ **do**
5: $\quad\quad$ **foreach** maximally specific $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ such that $\sigma_{\mathtt{b}} = \mathrm{mgu}(\mathtt{b}, \mathtt{B})$ exists **do**
6: $\quad\quad\quad$ **foreach** $\mathtt{h} = \mathtt{H}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}} \in G_{\Sigma}(\mathtt{H}\sigma_{\mathtt{b}})$ such that $\mathtt{h} \in S_{t+1}$ **and** $\mathtt{o_{t-1}}$ unifies with $\mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}}$ **do**
7: $\quad\quad\quad\quad \xi(\mathrm{cl}) := \xi(\mathrm{cl}) + \big(\alpha_t(\mathtt{b}) \cdot p \cdot \beta_{t+1}(\mathtt{h}) / P(O \mid M)\big) \boxed{\cdot \mu(\mathtt{h} \mid \mathtt{H}\sigma_{\mathtt{b}}) \cdot \mu(\mathtt{o_{t-1}} \mid \mathtt{O}\sigma_{\mathtt{b}}\sigma_{\mathtt{h}})}$
11: */* estimate of new transition probabilities */*
11: **foreach** $\mathrm{cl} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta$ **do**
12: $\quad \gamma(\mathtt{B}) := 0$
13: $\quad$ **foreach** $\mathrm{cl}' \equiv p' : \mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B} \in \Delta$ **do**
14: $\quad\quad \gamma(\mathtt{B}) := \gamma(\mathtt{B}) + \xi(\mathrm{cl}')$
12: $\quad \overline{p} = \xi(\mathrm{cl}) / \gamma(\mathtt{B})$

where equation (12) can be found in line 7. To estimate the selection probabilities, recall that we used a naïve Bayes scheme to define the selection probability implicitly. Therefore, the estimated probability for a domain element $d \in D$ for some domain D is the ratio between the number of times d is selected and the number of times any $d' \in D$ is selected. The procedure for computing the ξ-values can mainly be reused.

Altogether, our EM scheme is: While not converged, (1) estimate the abstract transition probabilities and (2) the selection probabilities. Since our algorithm is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to [20], it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. The computational complexity of parameter reestimation (per iteration) is $\mathcal{O}(4 \cdot k \cdot \alpha + d)$ where k is the number of sequences, α is the complexity of computing the α-values (see above), and d is the sum over the sizes of the domains associated to the predicates.