# Logical Hidden Markov Models

**Kristian Kersting** kersting@informatik.uni-freiburg.de
**Luc De Raedt** deraedt@informatik.uni-freiburg.de
*Machine Learning Lab, Computer Science Department, University of Freiburg*
*Georges-Koehler-Allee, Building 079, 79110 Freiburg, Germany*

**Tapani Raiko** tapani.raiko@hut.fi
*Laboratory of Computer and Information Science, Helsinki University of Technology*
*P.O. Box 5400 Helsinki, FIN-02015 HUT, Finland*

## Abstract

Logical hidden Markov models (LOHMMs) upgrade traditional hidden Markov models (HMMs) to deal with sequences of structured symbols in the form of logical atoms, rather than flat characters.

This note formally introduces LOHMMs and presents solutions to the three central inference problems for LOHMMs: evaluation, most likely hidden state sequence and parameter estimation. The resulting representation and algorithms are experimentally evaluated on problems from the domain of bioinformatics.

## 1. Introduction

Hidden Markov models (Rabiner & Juang, 1986) (HMMs) are extremely popular for analyzing sequential data. Application areas 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. Yet, in many applications the symbols occurring in sequences are structured. Consider, e.g., sequences of UNIX commands, which may have parameters such as `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ...Thus, commands are essentially structured. Tasks that have been considered for UNIX command sequences include the prediction of the next command in the sequence (Davison & Hirsh, 1998), the classification of a command sequence in a user category (Korvemaker & Greiner, 2000; Jacobs & Blockeel, 2001), and anomaly detection (Lane, 1999). Traditional HMMs cannot easily deal with this type of structured sequences. Indeed, applying 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 a serious information loss; the latter leads to a combinatorial explosion in the number of symbols and parameters of the HMM 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 as studied in the fields of inductive logic programming (Muggleton & De Raedt, 1994) and multi-relational learning (Džeroski & Lavrač, 2001). In this paper, we propose an (inductive) logic programming framework, Logical HMMs (LOHMMs), that upgrades HMMs to deal with structure. 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 $\texttt{emacs(lohmms.tex)}, \texttt{ls}, \texttt{latex(lohmms.tex)}, \ldots$ There are two important motivations for using logical atoms at the symbol level. First, *variables* in the atoms allow one to make abstraction of specific symbols. E.g., the logical atom $\texttt{emacs(X, tex)}$ represents all files $\texttt{X}$ that a LaTeXuser $\texttt{tex}$ could edit using $\texttt{emacs}$. Second, *unification* allows one to share information among states. E.g., the sequence $\texttt{emacs(X, tex)}, \texttt{latex(X, tex)}$ denotes that the same file is used as an argument for both Emacs and LaTeX.

The paper is organized as follows. After reviewing the logical preliminaries in Section 2, we introduce LOHMMs and define their semantics in Section 3; in Section 4, we upgrade the basic inference algorithms for HMMs for use in LOHMMs; we investigate the benefits of LOHMMs in Section 5: we show that LOHMMs are strictly more expressive than HMMs, that they can be — by design — at least an order of magnitude smaller than their corresponding propositional instantiations, and that unification can yield models which better fit the data. In Section 6, we empirically investigate the benefits of LOHMMs on real world data. Before concluding, we discuss related work in Section 7. Proofs of all theorems can be found in the Appendix.

## 2. Logical Preliminaries

A *first-order alphabet* $\Sigma$ is a set of relation symbols $\texttt{r}$ with arity $m \geq 0$, written $\texttt{r}/m$, and a set of functor symbols $\texttt{f}$ with arity $n \geq 0$, written $\texttt{f}/n$. If $n = 0$ then $\texttt{f}$ is called a constant, if $m = 0$ then $\texttt{p}$ is called a propositional variable. (We assume that at least one constant is given.) An *atom* $\texttt{r}(\texttt{t}_1, \ldots, \texttt{t}_n)$ is a relation symbol $\texttt{r}$ followed by a bracketed $n$-tuple of terms $\texttt{t}_i$. A *term* $\texttt{T}$ is a variable $\texttt{V}$ or a functor symbol $\texttt{f}(\texttt{t}_1, \ldots, \texttt{t}_k)$ immediately followed by a bracketed $k$-tuple of terms $\texttt{t}_i$. Variables will be written in upper-case, and constant, functor and predicate symbols lower-case. The symbol _ will denote anonymous variables which are read and treated as distinct, new variables each time they are encountered. An *iterative clause* is a formula of the form $\texttt{H} \leftarrow \texttt{B}$ where $\texttt{H}$ (called *head*) and $\texttt{B}$ (called *body*) are logical atoms. A substitution $\theta = \{\texttt{V}_1/\texttt{t}_1, \ldots, \texttt{V}_n/\texttt{t}_n\}$, e.g. $\{\texttt{X/tex}\}$, is an assignment of terms $\texttt{t}_i$ to variables $\texttt{V}_i$. Applying a substitution $\sigma$ to a term, atom or clause $\texttt{e}$ yields the instantiated term, atom, or clause $\texttt{e}\sigma$ where all occurrences of the variables $\texttt{V}_i$ are simultaneously replaced by the term $\texttt{t}_i$, e.g. $\texttt{ls(X)} \leftarrow \texttt{emacs(F, X)}\{\texttt{X/tex}\}$ yields $\texttt{ls(tex)} \leftarrow \texttt{emacs(F, tex)}$. A substitution $\sigma$ is called a *unifier* for a finite set $S$ of atoms if $S\sigma$ is singleton. A unifier $\theta$ for $S$ is called a *most general unifier* (MGU) for $S$ if, for each unifier $\sigma$ of $S$, there exists a substitution $\gamma$ such that $\sigma = \theta\gamma$. A term, atom or clause $\texttt{E}$ is called *ground* when it contains no variables, i.e., $vars(\texttt{E}) = \emptyset$. The *Herbrand base* of $\Sigma$, denoted as $\mathrm{hb}_\Sigma$, is the set of all ground atoms constructed with the predicate and functor symbols in $\Sigma$. The set $G_\Sigma(\texttt{A})$ of an atom $\texttt{A}$ consists of all ground atoms $\texttt{A}\theta$ that belong to $\mathrm{hb}_\Sigma$.

## 3. Logical Hidden Markov Models

The logical component of a traditional HMM corresponds to a *Mealy machine* (Hopcroft & Ullman, 1979), i.e., a finite state machine where the output symbols are associated with transitions. This is essentially a propositional representation because the symbols used to represent states and output symbols are flat, i.e. not structured. The key idea underlying LOHMMs is to replace these flat symbols by abstract symbols. An abstract symbol $\texttt{A}$ is —

by definition — a logical atom. It is abstract in that it represents the set of all ground, i.e., variable-free atoms of $\mathtt{A}$ over the alphabet $\Sigma$, denoted by $G_\Sigma(\mathtt{A})$. Ground atoms then play the role of the traditional symbols used in a HMMs.

**Example 1** *Consider the alphabet* $\Sigma_1$ *which has as constant symbols* `tex`*,* `dvi`*,* `hmm1`*, and* `lohmm1`*, and as relation symbols* `emacs/2`*,* `ls/1`*,* `xdvi/1`*,* `latex/2`*. Then the atom* $\mathtt{emacs(File, tex)}$ *represents the set* $\{\mathtt{emacs(hmm1, tex)}, \mathtt{emacs(lohmm1, tex)}\}$*. We assume that the alphabet is typed to avoid useless instantiations such as* $\mathtt{emacs(tex, tex)}$*).*

The use of atoms instead of flat symbols allows us to analyze logical and structured sequences such as $\mathtt{emacs(hmm1, tex)}, \mathtt{latex(hmm1, tex)}, \mathtt{xdvi(hmm1, dvi)}$.

**Definition 1** Abstract transition *are expressions 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. All variables are implicitly assumed to be universally quantified, i.e., the scope of variables is a single abstract transition.*

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 with probability $p$ to one of the states in $G_\Sigma(\mathtt{H}\theta_\mathtt{B})$, say $\mathtt{H}\theta_\mathtt{B}\theta_\mathtt{H}$, while emitting a symbol 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* $c \equiv 0.8 : \mathtt{xdvi(File, dvi)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, tex)}$*. In general* $\mathtt{H}$*,* $\mathtt{B}$ *and* $\mathtt{O}$ *do not have to share the same predicate. This is only due to the nature of our running example. Assume now that we are in state* $\mathtt{latex(hmm1, tex)}$*, i.e.* $\theta_\mathtt{B} = \{\mathtt{File/hmm1}\}$*. Then c specifies that there is a probability of 0.8 that the next state will be in* $G_{\Sigma_1}(\mathtt{xdvi(hmm1, dvi)}) = \{\mathtt{xdvi(hmm1, dvi)}\}$ *( i.e., the probability is* $0.8$ *that the next state will be* $\mathtt{xdvi(hmm1, dvi)}$*), and that one of the symbols in* $G_{\Sigma_1}(\mathtt{latex(hmm1)}) = \{\mathtt{latex(hmm1)}\}$ *( i.e.,* $\mathtt{latex(hmm1)}$*) will be emitted. Abstract states might also be more complex such as* $\mathtt{latex(file(FileStem, FileExtension), User)}$

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

**Definition 2** *The selection distribution* $\mu$ *specifies for each abstract state and observation symbol* $\mathtt{A}$ *over the alphabet* $\Sigma$ *a distribution* $\mu(\cdot \mid \mathtt{A})$ *over* $G_\Sigma(\mathtt{A})$*.*

To continue our example, 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 is $\mathtt{emacs(hmm1, tex)}$ and of 0.48 that it is $\mathtt{emacs(lohmm1, tex)}$.

Taking $\mu$ into account, 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 $\mu$, any probabilistic representation can - in principle - be used, e.g. a Bayesian network or a Markov chain. Throughout the remainder of the present paper, however, we will use a *naïve Bayes* approach. More precisely, we associate to each argument of a relation $\mathtt{r}/m$ a finite domain $\mathrm{D}_i^{\mathtt{r}/m}$ of constants and a probability distribution $P_i^{\mathtt{r}/m}$ over $\mathrm{D}_i^{\mathtt{r}/m}$. Let $\mathrm{vars}(\mathtt{A}) = \{\mathtt{V}_1, \ldots, \mathtt{V}_l\}$ be the variables occurring in an atom $\mathtt{A}$ over $\mathtt{r}/m$, and let $\sigma = \{\mathtt{V}_1/\mathtt{s}_1, \ldots \mathtt{V}_l/\mathtt{s}_l\}$ be a substitution grounding $\mathtt{A}$. Each $\mathtt{V}_j$ is then considered a random variable over the domain $D^{\mathtt{r}/m}_{\mathrm{arg}(\mathtt{V}_j)}$ of the argument $\mathrm{arg}(\mathtt{V}_j)$ it appears first in. Then, $\mu(\mathtt{A}\sigma \mid \mathtt{A}) = \prod_{j=1}^{l} P^{\mathtt{r}/m}_{\mathrm{arg}(\mathtt{V}_j)}(\mathtt{s}_j)$. E.g. $\mu(\mathtt{emacs}(\mathtt{hmm1}, \mathtt{tex}) \mid \mathtt{emacs}(\mathtt{F}, \mathtt{E}))$, is computed as the product of $P_1^{\mathtt{emacs}/2}(\mathtt{hmm1})$ and $P_2^{\mathtt{emacs}/2}(\mathtt{tex})$.

Thus far the semantics of a single abstract transition has been defined. A LOHMM usually consists of multiple abstract transitions and this creates a further complication.

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

There are essentially two ways to deal with this situation. On the one hand, one might want to combine and normalize the two transitions and assign a probability of $\frac{2}{3}$ respectively $\frac{1}{3}$ . On the other hand, one might want to have only one rule firing. In this paper, we chose the latter option because it allows us to consider transitions more independently, it simplifies learning, and it yields locally interpretable models. We employ the subsumption (or generality) relation among the $\mathtt{B}$-parts of the two abstract transitions. Indeed, the $\mathtt{B}$-part of the first transition $\mathtt{B}_1 = \mathtt{emacs}(\mathtt{File}, \mathtt{tex})$ is more specific than that of the second transition $\mathtt{B}_2 = \mathtt{emacs}(\mathtt{File}, \mathtt{User})$ because there exists a substitution $\theta = \{\mathtt{User}/\mathtt{tex}\}$ such that $\mathtt{B}_2\theta = \mathtt{B}_1$, i.e., $\mathtt{B}_2$ subsumes $\mathtt{B}_1$. Therefore $G_{\Sigma_1}(\mathtt{B}_1) \subseteq G_{\Sigma_1}(\mathtt{B}_2)$ and the first transition can be regarded as more informative than the second one. It should therefore be preferred over the second one when starting from $\mathtt{emacs}(\mathtt{hmm1}, \mathtt{tex})$. 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 and is akin to Katz (1987)'s *back-off* $n$-gram models. In back-off $n$-gram models, the most detailed model that is deemed to provide sufficiently reliable information about the current context is used. That is, if one encounters an $n$-gram that is not sufficiently reliable, then back-off to use an $(n-1)$-gram; if that is not reliable either then back-off to level $n-2$, etc.

The conflict resolution strategy will work properly provided that the bodies of all maximally specific transitions (matching a given state) represent the same abstract state. This can be enforced by requiring the *generality* relation over the $\mathtt{B}$-parts to be closed under the *greatest lower bound* (glb) for each predicate, i.e., for each pair $\mathtt{B}_1, \mathtt{B}_2$ of bodies, such that

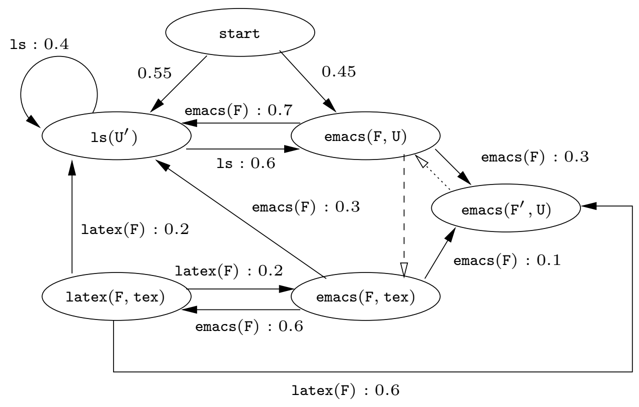


Figure 1: A logical hidden Markov model.

$\theta = \text{mgu}(\mathtt{B}_1, \mathtt{B}_2)$ exists, there is another body $\mathtt{B}$ (called lower bound) which subsumes $\mathtt{B}_1\theta$ (therefore also $\mathtt{B}_2\theta$) and is subsumed by $\mathtt{B}_1, \mathtt{B}_2$, and if there is any other lower bound then it is subsumed by $\mathtt{B}$. E.g., if the body of the second abstract transition in our example is $\mathtt{emacs(hmm1, User)}$ then the set of abstract transitions would not be closed under glb.

Finally, in order to specify a prior distribution over states, we assume a finite set $\Upsilon$ of clauses of the form $p : \mathtt{H} \leftarrow \mathtt{start}$ using a distinguished $\mathtt{start}$ symbol such that $p$ is the probability of the LOHMM to start in a state of $G_\Sigma(\mathtt{H})$.

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

**Definition 3** *A logical hidden Markov model (LOHMM) is a tuple* $(\Sigma, \mu, \Delta, \Upsilon)$ *where* $\Sigma$ *is a logical alphabet,* $\mu$ *a selection probability over* $\Sigma$*,* $\Delta$ *is a set of abstract transitions, and* $\Upsilon$ *is a set of abstract transitions encoding a prior distribution. Let* $\mathbf{B}$ *be the set of all atoms that occur as body parts of transitions in* $\Delta$*. We assume* $\mathbf{B}$ *to be closed under glb and require*

$$\forall \mathtt{B} \in \mathbf{B} : \sum\nolimits_{p:\mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta} p = 1.0 \qquad (2)$$

*and that the probabilities* $p$ *of clauses in* $\Upsilon$ *sum up to* $1.0$ *.*

HMMs are a special cases of LOHMMs in which $\Sigma$ contains only relation symbols of arity zero and the selection probability is irrelevant. Thus, LOHMMs directly generalize HMMs.

LOHMMs can also be represented graphically. Figure 1 contains an example. The underlying language $\Sigma_2$ consists of $\Sigma_1$ together with the constant symbol $\mathtt{other}$ which denotes a user that does not employ LaTeX. In this graphical notation, nodes represent abstract states and *black tipped arrows* denote abstract transitions. *White tipped arrows* are used to represent meta knowledge. More precisely, *white tipped, dashed arrows* represent the generality or subsumption ordering between abstract states. If we follow a transition to an abstract state with an outgoing *white tipped, dotted arrow* then this dotted arrow will always be followed. Dotted arrows are needed because the same abstract state can occur under different circumstances. Consider the transition $p : \mathtt{latex(File', User')} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, User)}$. Even though the atoms in the head and body of the transition are syntactically different they

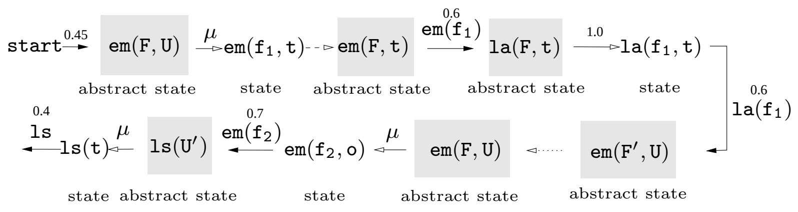


Figure 2: Generating the observation sequence emacs(hmm1), latex(hmm1), emacs(lohmm1), ls by the LOHMM in Figure 1. The command emacs is abbreviated by em, $\mathtt{f}_1$ denotes the filename hmm1, $\mathtt{f}_2$ represents lohmm1, t denotes a tex user, and o some other user. White tipped solid arrows indicate selections.

represent the same abstract state. To accurately represent the meaning of this transition we cannot use a black tipped arrow from latex(File, User) to itself, because this would actually represent the abstract transition $p : \mathtt{latex(File, User)} \xleftarrow{\mathtt{latex(File)}} \mathtt{latex(File, User)}$.

Furthermore, the graphical representation clarifies that LOHMMs are generative models. Let us explain how the model in Figure 1 would generate the observation sequence emacs(hmm1), latex(hmm1), emacs(lohmm1), ls (cf. Figure 2). It chooses an initial abstract state, say emacs(F, U). Since both variables F and U are uninstantiated, the model samples the state emacs(hmm1, tex) from $G_{\Sigma_2}$ using $\mu$. As indicated by the dashed arrow, emacs(F, tex) is more specific than emacs(F, U). Moreover, emacs(hmm1, tex) matches emacs(F, tex). Thus, the model enters emacs(F, tex). Since the value of F was already instantiated in the previous abstract state, emacs(hmm1, tex) is sampled with probability 1.0. Now, the model goes over to latex(F, tex), emitting emacs(hmm1) because the abstract observation emacs(F) is already fully instantiated. Again, since $F$ was already instantiated, latex(hmm1, tex) is sampled with probability 1.0. Next, we move on to emacs(F′, U), emitting latex(hmm1). Variables F′ and U in emacs(F′, U) were not yet bound; so, values, say lohmm1 and others, are sampled from $\mu$. The dotted arrow brings us back to emacs(F, U). Because variables are implicitly universally quantified in abstract transitions, the scope of variables is restricted to single abstract transitions. In turn, $F$ is treated as a distinct, new variable, and is automatically unified with F′, which is bound to lohmm1. In contrast, variable U is already instantiated. Emitting emacs(lohmm1), the model makes a transition to ls(U′). Assume that it samples tex for U′. Then, it remains in ls(U′) with probability 0.4 . Considering all possible samples, allows one to prove the following theorem.

**Theorem 1 (Semantics)** *A logical hidden Markov model over a language $\Sigma$ defines a discrete time stochastic process, i.e., a sequence of random variables $\langle X_t \rangle_{t=1,2,\ldots}$, where the domain of $X_t$ is* $\mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$*. The induced probability measure over the Cartesian product* $\bigotimes_t \mathrm{hb}(\Sigma) \times \mathrm{hb}(\Sigma)$ *exists and is unique for each $t > 0$ and in the limit $t \to \infty$.*

Before concluding this section, let us address some design choices underlying LOHMMs.

First, LOHMMs have been introduced as Mealy machines, i.e., output symbols are associated with transitions. Mealy machines fit our logical setting quite intuitively as they directly encode the conditional probability $P(\mathtt{O}, \mathtt{S}'|\mathtt{S})$ of making a transition from S to S′

emitting an observation $\mathtt{O}$. Logical hidden Markov models define this distribution as

$$P(\mathtt{O},\mathtt{S}'|\mathtt{S}) = \sum\nolimits_{p:\mathtt{H}\xleftarrow{\mathtt{O}'}\mathtt{B}} p \;\cdot\; \mu(\mathtt{S}' \mid \mathtt{H}\sigma_\mathtt{B}) \;\cdot\; \mu(\mathtt{O} \mid \mathtt{O}'\sigma_\mathtt{B}\sigma_\mathtt{H})$$

where the sum runs over all abstract transitions $\mathtt{H} \xleftarrow{\mathtt{O}'} \mathtt{B}$ such that $\mathtt{B}$ is most specific for $\mathtt{S}$. Observations correspond to (partially) observed proof steps and, hence, provide information shared among heads and bodies of abstract transitions. In contrast, HMMs are usually introduced as *Moore* machines. Here, output symbols are associated with states implicitly assuming $\mathtt{O}$ and $\mathtt{S}'$ to be independent. Thus, $P(\mathtt{O},\mathtt{S}' \mid \mathtt{S})$ factorizes into $P(\mathtt{O} \mid \mathtt{S}) \cdot P(\mathtt{S}' \mid \mathtt{S})$. This makes it more difficult to observe information shared among heads and bodies. In turn, Moore-LOHMMs are less intuitive and harder to understand. For a more detailed discussion of the issue, we refer to Appendix B where we essentially show that – as in the propositional case – Mealy- and Moore-LOHMMs are equivalent.

Second, the naïve Bayes approach for the selection distribution reduces the model complexity at the expense of a lower expressivity: functors are neglected and variables are treated independently. Adapting more expressive approaches is an interesting future line of research. For instance, *Bayesian networks* allow one to represent *factorial HMMs* (Ghahramani & Jordan, 1997). Factorial HMMs can be viewed as a special type of LOHMMs, where the hidden states are 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. $\mu$ of the $i$-th argument is conditioned on the $i+k$-th argument. *Markov chains* allow one to sample compound terms of variable, finite depth such as $\mathtt{s(s(s(0)))}$ and to model e.g. misspelled filenames. This is akin to the idea of *generalized HMMs* (Kulp et al., 1996) in which each node may output a finite sequence of symbols rather than a single symbol.

Finally, LOHMMs – as introduced in the present paper – specify a probability distribution over all sequences of a given length. Reconsider the LOHMM in Figure 1. Already the probabilities of all observation sequences of length 1, i.e., `ls`, `emacs(hmm1)`, and `emacs(lohmm1)`) sum up to 1. More precisely, for each $t > 0$ it holds that $\sum_{x_1,\ldots,x_t} P(X_1 = x_1,\ldots,X_t = x_t) = 1.0$ . In order to model a distribution over sequences of variable length, i.e., $\sum_{t>0}\sum_{x_1,\ldots,x_t} P(X_1 = x_1,\ldots,X_t = x_t) = 1.0$ we may add a distinguished `end` state. The `end` state is absorbing in that whenever the model makes a transition into this state, it terminates the observation sequence generated.

## 4. Three Inference Problems for LOHMMs

As for HMMs, three inference problems are of interest. Let $M$ be a LOHMM and let $\mathsf{O} = \mathtt{O}_1,\mathtt{O}_2,\ldots,\mathtt{O}_T$, $T > 0$, be a finite sequence of ground observations:

**(1) Evaluation:** Determine the probability $P(\mathsf{O} \mid M)$ that sequence $\mathsf{O}$ was generated by the model $M$.

**(2) Most likely state sequence:** Determine the hidden state sequence $\mathsf{S}^*$ that has most likely produced the observation sequence $\mathsf{O}$, i.e. $\mathsf{S}^* = \arg\max_\mathsf{S} P(\mathsf{S} \mid \mathsf{O}, M)$ .

**(3) Parameter estimation:** Given a set $\mathbf{O} = \{\mathsf{O}_1,\ldots,\mathsf{O}_k\}$ of observation sequences, determine the most likely parameters $\lambda^*$ for the abstract transitions and the selection distribution of $M$, i.e. $\lambda^* = \arg\max_\lambda P(\mathbf{O} \mid \lambda)$ .

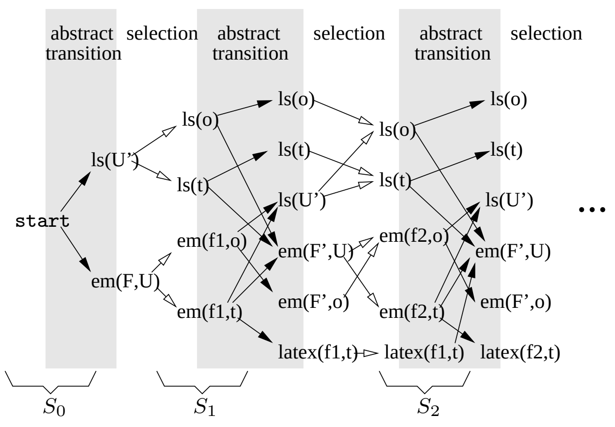


Figure 3: Trellis induced by the LOHMM in Figure 1. The sets of reachable states at time $0, 1, \ldots$ are denoted by $S_0, S_1, \ldots$ In contrast with HMMs, there is an additional layer where the states are sampled from abstract states.

We will now address each of these problems in turn by upgrading the existing solutions for HMMs. This will be realized by computing a grounded trellis as in Figure 3. The possible ground successor states of any given state are computed by first selecting the applicable abstract transitions and then applying the selection probabilities (while taking into account the substitutions) to ground the resulting states. This two-step factorization is coalesced into one step for HMMs.

To **evaluate** $\mathtt{O}$, consider the probability of the partial observation sequence $\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t$ and (ground) state $\mathtt{S}$ at time $t$, $0 < t \leq T$, given the model $M = (\Sigma, \mu, \Delta, \Upsilon)$

$$\alpha_t(\mathtt{S}) := P(\mathtt{O}_1, \mathtt{O}_2, \ldots, \mathtt{O}_t, q_t = \mathtt{S} \mid M)$$

where $q_t = \mathtt{S}$ denotes that the system is in state $\mathtt{S}$ at time $t$. As for HMMs, $\alpha_t(\mathtt{S})$ can be computed using a dynamic programming approach. For $t = 0$, we set $\alpha_0(\mathtt{S}) = P(q_0 = \mathtt{S} \mid M)$ , i.e., $\alpha_0(\mathtt{S})$ is the probability of starting in state $\mathtt{S}$ and, for $t > 0$, we compute $\alpha_t(\mathtt{S})$ based on $\alpha_{t-1}(\mathtt{S}')$:

```
1:  $S_0 := \{\mathtt{start}\}$                 /* initialize the set of reachable states*/
2:  for $t = 1, 2, \ldots, T$ do
3:      $S_t = \emptyset$                        /* initialize the set of reachable states at clock t*/
4:      foreach $\mathtt{S} \in S_{t-1}$ do
5:          $\boxed{\textbf{foreach} \text{ maximally specific } p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \cup \Upsilon \text{ s.t. } \sigma_{\mathtt{B}} = \mathrm{mgu}(\mathtt{S}, \mathtt{B}) \text{ exists } \textbf{do}}$
6:              $\boxed{\textbf{foreach } \mathtt{S}' = \mathtt{H}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \in G_\Sigma(\mathtt{H}\sigma_{\mathtt{B}}) \text{ s.t. } \mathtt{O}_{t-1} \text{ unifies with } \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \textbf{ do}}$
7:                  if $\mathtt{S}' \notin S_t$ then
8:                      $S_t := S_t \cup \{\mathtt{S}'\}$
9:                      $\alpha_t(\mathtt{S}') := 0.0$
10:                 $\alpha_t(\mathtt{S}') := \alpha_t(\mathtt{S}') + \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \boxed{\mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}$
11: return $P(\mathtt{O} \mid M) = \sum_{\mathtt{S} \in S_T} \alpha_T(\mathtt{S})$
```

where we assume for the sake of simplicity $\mathtt{O} \equiv \mathtt{start}$ for each abstract transition $p : \mathtt{H} \leftarrow \mathtt{start} \in \Upsilon$. Furthermore, the boxed parts specify all the differences to the HMM formula: unification and $\mu$ are taken into account.

Clearly, as for HMMs $P(\mathtt{O} \mid M) = \sum_{\mathtt{S} \in S_T} \alpha_T(\mathtt{S})$ holds. The computational complexity of this *forward procedure* is $\mathcal{O}(T \cdot s \cdot (|\mathbf{B}| + o \cdot g)) = \mathcal{O}(T \cdot s^2)$ where $s = \max_{t=1,2,\ldots,T} |S_t|$ , $o$ 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

$$\beta_t(\mathtt{S}) = P(\mathtt{O}_{t+1}, \mathtt{O}_{t+2}, \ldots, \mathtt{O}_T \mid q_t = \mathtt{S}, M) \ .$$

This will be useful for solving Problem **(3)**.

Having a forward procedure, it is straightforward to adapt the Viterbi algorithm as a solution to Problem **(2)**, i.e., for computing the **most likely state sequence**. Let $\delta_t(\mathtt{S})$ denote the highest probability along a single path at time $t$ which accounts for the first $t$ observations and ends in state $\mathtt{S}$, i.e.,

$$\delta_t(\mathtt{S}) = \max_{\mathtt{S}_0, \mathtt{S}_1, \ldots, \mathtt{S}_{t-1}} P(\mathtt{S}_0, \mathtt{S}_1, \ldots, \mathtt{S}_{t-1}, \mathtt{S}_t = \mathtt{S}, O_1, \ldots, O_{t-1} | M) \ .$$

The procedure for finding the most likely state sequence basically follows the forward procedure. Instead of summing over all ground transition probabilities in line 10, we maximize over them. More precisely, we proceed as follows:

```
1: $S_0 := \{\mathtt{start}\}$          /* initialize the set of reachable states*/
2: **for** $t = 1, 2, \ldots, T$ **do**
3:     $S_t = \emptyset$          /* initialize the set of reachable states at clock t*/
4:     **foreach** $\mathtt{S} \in S_{t-1}$ **do**
5:         **foreach** maximally specific $p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \cup \Upsilon$ s.t. $\sigma_\mathtt{B} = \mathrm{mgu}(\mathtt{S}, \mathtt{B})$ exists **do**
6:             **foreach** $\mathtt{S}' = \mathtt{H}\sigma_\mathtt{B}\sigma_\mathtt{H} \in G_\Sigma(\mathtt{H}\sigma_\mathtt{B})$ s.t. $\mathtt{O}_{t-1}$ unifies with $\mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H}$ **do**
7:                 **if** $\mathtt{S}' \notin S_t$ **then**
8:                     $S_t := S_t \cup \{\mathtt{S}'\}$
9:                     $\delta_t(\mathtt{S}, \mathtt{S}') := 0.0$
10:                $\delta_t(\mathtt{S}, \mathtt{S}') := \delta_t(\mathtt{S}, \mathtt{S}') + \delta_{t-1}(\mathtt{S}) \cdot p \ \cdot \ \mu(\mathtt{S}' \mid \mathtt{H}\sigma_\mathtt{B}) \ \cdot \ \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H})$
11:    **foreach** $\mathtt{S}' \in S_t$ **do**
12:        $\delta_t(\mathtt{S}') = \max_{\mathtt{S} \in S_{t-1}} \delta_t(\mathtt{S}, \mathtt{S}')$
13:        $\psi_t(\mathtt{S}') = \arg\max_{\mathtt{S} \in S_{t-1}} \psi_t(\mathtt{S}, \mathtt{S}')$
```

Here, $\delta_t(\mathtt{S}, \mathtt{S}')$ stores the probability of making a transition from $\mathtt{S}$ to $\mathtt{S}'$ and $\psi_t(\mathtt{S}')$ (with $\psi_1(S) = \mathtt{start}$ for all states $S$) keeps track of the state maximizing the probability along a single path at time $t$ which accounts for the first $t$ observations and ends in state $\mathtt{S}'$. The most likely hidden state sequence $\mathtt{S}^*$ can now be computed as

$$\begin{aligned} \mathtt{S}^*_{T+1} &= \arg \max_{\mathtt{S} \in S_{T+1}} \delta_{T+1}(\mathtt{S}) \\ \text{and } \mathtt{S}^*_t &= \psi_t(\mathtt{S}^*_{t+1}) \text{ for } t = T, T-1, \ldots, 1 \ . \end{aligned}$$

One can also consider problem **(2)** on a more abstract level. Instead of considering all contributions of different abstract transitions $\mathtt{T}$ to a single ground transition from state $\mathtt{S}$

to state $\mathtt{S}'$ in line 10, one might also consider the most likely abstract transition only. This is realized by replacing line 10 in the forward procedure with

$$\alpha_t(\mathtt{S}') := \max(\alpha_t(\mathtt{S}'), \alpha_{t-1}(\mathtt{S}) \cdot p \cdot \ \mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}}))\ .$$

This solves the problem of finding the **(2′) most likely state and abstract transition sequence**:

> Determine the sequence of states and abstract transitions $\mathsf{GT}^* = S_0, \mathtt{T}_0, S_1, \mathtt{T}_1, \mathtt{S}_2, \ldots, \mathtt{S}_\mathtt{T}, \mathtt{T}_T, \mathtt{S}_{\mathtt{T}+1}$ where there exists substitutions $\theta_i$ with $S_{i+1} \leftarrow S_i \equiv \mathtt{T}_i\,\theta_i$ that has most likely produced the observation sequence $\mathsf{O}$, i.e. $\mathsf{GT}^* = \arg\max_{\mathsf{GT}} P(\mathsf{GT} \mid \mathsf{O}, M)$ .

Thus, logical hidden Markov models also pose new types of inference problems.

For **parameter estimation**, we have to estimate the maximum likelihood transition probabilities and selection distributions. To estimate the former, we upgrade the well-known *Baum-Welch* algorithm (Baum, 1972) for estimating the maximum likelihood parameters of HMMs and probabilistic context-free grammars.

For HMMs, the Baum-Welch algorithm computes the improved estimate $\overline{p}$ of the transition probability of some (ground) transition $\mathtt{T} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B}$ by taking the ratio

$$\overline{p} = \frac{\xi(\mathtt{T})}{\sum_{\mathtt{H}' \xleftarrow{\mathtt{O}'} \mathtt{B} \in \Delta \cup \Upsilon} \xi(\mathtt{T}')} \tag{3}$$

between the expected number $\xi(\mathtt{T})$ of times of making the transitions $\mathtt{T}$ at any time given the model $M$ and an observation sequence $\mathsf{O}$, and the total number of times a transitions is made from $\mathtt{B}$ at any time given $M$ and $\mathsf{O}$.

Basically the same applies when $\mathtt{T}$ is an abstract transition. However, we have to be a little bit more careful because we have no direct access to $\xi(\mathtt{T})$. Let $\xi_t(\mathrm{gcl}, \mathtt{T})$ be the probability of following the abstract transition $\mathtt{T}$ via its ground instance $\mathrm{gcl} \equiv p : \mathtt{GH} \xleftarrow{\mathtt{GO}} \mathtt{GB}$ at time $t$, i.e.,

$$\xi_t(\mathrm{gcl}, \mathtt{T}) = \frac{\alpha_t(\mathtt{GB}) \cdot p \cdot \beta_{t+1}(\mathtt{GH})}{P(\mathsf{O} \mid M)} \cdot \boxed{\mu(\mathtt{GH} \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}\ , \tag{4}$$

where $\sigma_{\mathtt{B}}, \sigma_{\mathtt{H}}$ are as in the forward procedure (see above) and $P(\mathsf{O} \mid M)$ is the probability that the model generated the sequence $\mathsf{O}$. Again, the boxed terms constitute the main difference to the corresponding HMM formula. In order to apply Equation (3) to compute improved estimates of probabilities associated with abstract transitions, we set

$$\xi(\mathtt{T}) = \sum_{t=1}^{T} \xi_t(\mathtt{T}) = \sum_{t=1}^{T} \sum_{\mathrm{gcl}} \xi_t(\mathrm{gcl}, \mathtt{T})$$

where the inner sum runs over all ground instances of $\mathtt{T}$.

This leads to the following re-estimation method, where we assume that the sets $S_i$ of reachable states are reused from the computations of the $\alpha$- and $\beta$-values:

```
1: /* initialization of expected counts */
2: foreach $\mathtt{T} \in \Delta \cup \Upsilon$ do
3:     $\xi(\mathtt{T}) := m$  /* or 0 if not using pseudocounts */
4: /* compute expected counts */
5: for $t = 0, 1, \ldots, T$ do
6:     foreach $\mathtt{S} \in S_t$ do
7:         foreach max. specific $\mathtt{T} \equiv p : \mathtt{H} \xleftarrow{\mathtt{O}} \mathtt{B} \in \Delta \cup \Upsilon$ s.t. $\sigma_{\mathtt{B}} = \mathrm{mgu}(\mathtt{S}, \mathtt{B})$ exists do
8:             foreach $\mathtt{S}' = \mathtt{H}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}} \in G_{\Sigma}(\mathtt{H}\sigma_{\mathtt{B}})$ s.t. $\mathtt{S}' \in S_{t+1} \wedge \mathrm{mgu}(\mathtt{O}_t, \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})$ exists do
9:                 $\xi(\mathtt{T}) := \xi(\mathtt{T}) + \alpha_t(\mathtt{S}) \cdot p \cdot \beta_{t+1}(\mathtt{S}')/P(\mathtt{O} \mid M) \cdot \boxed{\mu(\mathtt{S}' \mid \mathtt{H}\sigma_{\mathtt{B}}) \cdot \mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_{\mathtt{B}}\sigma_{\mathtt{H}})}$
```

Here, equation (4) can be found in line 9. In line 3, we set pseudocounts as small sample-size regularizers. Other methods to avoid a biased underestimate of probabilities and even zero probabilities such as $m$-estimates, see e.g. (Mitchell, 1997), can be easily adapted.

To estimate the selection probabilities, recall that $\mu$ follows a naïve Bayes scheme. 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 $\xi$-values can thus be reused.

Altogether, the Baum-Welch algorithm works as follows: While not converged, (1) estimate the abstract transition probabilities, and (2) the selection probabilities. Since it is an instance of the EM algorithm, it increases the likelihood of the data with every update, and according to McLachlan and Krishnan (1997), it is guaranteed to reach a stationary point. All standard techniques to overcome limitations of EM algorithms are applicable. The computational complexity (per iteration) is $\mathcal{O}(k \cdot (\alpha + d)) = \mathcal{O}(k \cdot T \cdot s^2 + k \cdot d)$ where $k$ is the number of sequences, $\alpha$ is the complexity of computing the $\alpha$-values (see above), and $d$ is the sum over the sizes of domains associated to predicates. Recently, Kersting and Raiko (2005) combined the Baum-Welch algorithm with structure search for model selection of logical hidden Markov models using *inductive logic programming* (Muggleton & De Raedt, 1994) refinement operators. The refinement operators account for different abstraction levels which have to be explored.

## 5. Advantages of LOHMMs

In this section, we will investigate the benefits of LOHMMs: **(1)** LOHMMs are strictly more expressive than HMMs, and **(2)**, using abstraction, logical variables and unification can be beneficial. More specifically, with **(2)**, we will show that

**(B1)** LOHMMs can be — by design — smaller than their propositional instantiations, and

**(B2)** unification can yield better log-likelihood estimates.

### 5.1 On the Expressivity of LOHMMs

Whereas HMMs specify probability distributions over regular languages, LOHMMs specify probability distributions over more expressive languages.

**Theorem 2** *For any (consistent) probabilistic context-free grammar (PCFG) $G$ for some language $\mathcal{L}$ there exists a LOHMM $M$ s.t. $P_G(w) = P_M(w)$ for all $w \in \mathcal{L}$.*

The proof (see Appendix C) makes use of abstract states of unbounded 'depth'. More precisely, functors are used to implement a stack. Without functors, LOHMMs cannot encode PCFGs and, because the Herbrand base is finite, it can be proven that there always exists an equivalent HMM.

Furthermore, if functors are allowed, LOHMMs are strictly more expressive than PCFGs. They can specify probability distributions over some languages that are context-sensitive:

$$\begin{array}{rrcl}
1.0: & \texttt{stack(s(0),s(0))} & \leftarrow & \texttt{start} \\
0.8: & \texttt{stack(s(X),s(X))} & \xleftarrow{\texttt{a}} & \texttt{stack(X,X)} \\
0.2: & \texttt{unstack(s(X),s(X))} & \xleftarrow{\texttt{a}} & \texttt{stack(X,X)} \\
1.0: & \texttt{unstack(X,Y)} & \xleftarrow{\texttt{b}} & \texttt{unstack(s(X),Y)} \\
1.0: & \texttt{unstack(s(0),Y)} & \xleftarrow{\texttt{c}} & \texttt{unstack(s(0),s(Y))} \\
1.0: & \texttt{end} & \xleftarrow{\texttt{end}} & \texttt{unstack(s(0),s(0))}
\end{array}$$

The LOHMM defines a distribution over $\{a^n b^n c^n \mid n > 0\}$.

Finally, the use of logical variables also enables one to deal with *identifiers*. Identifiers are special types of constants that denote objects. Indeed, recall the UNIX command sequence `emacs lohmms.tex`, `ls`, `latex lohmms.tex`, ... from the introduction. The filename `lohmms.tex` is an identifier. Usually, the specific identifiers do not matter but rather the fact that the same object occurs multiple times in the sequence. LOHMMs can easily deal with identifiers by setting the selection probability $\mu$ to a constant for the arguments in which identifiers can occur. Unification then takes care of the necessary variable bindings.

## 5.2 Benefits of Abstraction through Variables and Unification

Reconsider the domain of UNIX command sequences. Unix users oftenly reuse a newly created directory in subsequent commands such as in `mkdir(vt100x)`, `cd(vt100x)`, `ls(vt100x)` . Unification should allow us to elegantly employ this information because it allows us to specify that, after observing the created directory, the model makes a transition into a state where the newly created directory is used:

$$p_1 : \texttt{cd(Dir,mkdir)} \leftarrow \texttt{mkdir(Dir,com)} \quad \text{and} \quad p_2 : \texttt{cd(\_,mkdir)} \leftarrow \texttt{mkdir(Dir,com)}$$

If the first transition is followed, the `cd` command will move to the newly created directory; if the second transition is followed, it is not specified which directory `cd` will move to. Thus, the LOHMM captures the reuse of created directories as an argument of future commands. Moreover, the LOHMM encodes the simplest possible case to show the benefits of unification. At any time, the observation sequence uniquely determines the state sequence, and functors are not used. Therefore, we left out the abstract output symbols associated with abstract transitions. In total, the LOHMM $U$, modelling the reuse of directories, consists of 542 parameters only but still covers more than 451000 (ground) states, see Appendix D for the complete model. The compression in the number of parameters supports **(B1)**.

To empirically investigate the benefits of unification, we compare $U$ with the variant $N$ of $U$ where no variables are shared, i.e., no unification is used such that for instance the first transition above is not allowed, see Appendix D. $N$ has 164 parameters less than $U$. We computed the following zero-one win function

$$f(\mathsf{O}) = \begin{cases} 1 & \text{if } \left[\log P_U(\mathsf{O}) - \log P_N(\mathsf{O})\right] > 0 \\ 0 & \text{otherwise} \end{cases}$$

leave-one-out cross-validated on Unix shell logs collected by Greenberg (1988). Overall, the data consists of 168 users of four groups: computer scientists, nonprogrammers, novices and others. About 300000 commands have been logged with an average of 110 sessions per user. We present here results for a subset of the data. We considered all computer scientist sessions in which at least a single `mkdir` command appears. These yield 283 logical sequences over in total 3286 ground atoms. The LOO win was 81.63%. Other LOO statistics are also in favor of $U$:

| | training | | test | |
|---|---|---|---|---|
| | $\log P(\mathbf{O})$ | $\log \frac{P_U(\mathbf{O})}{P_N(\mathbf{O})}$ | $\log P(\mathsf{O})$ | $\log \frac{P_U(\mathsf{O})}{P_N(\mathsf{O})}$ |
| $U$ | −11361.0 | 1795.3 | −42.8 | 7.91 |
| $N$ | −13157.0 | | −50.7 | |

Thus, although $U$ has 164 parameters more than $N$, it shows a better generalization performance. This result supports **(B2)**. A pattern often found in $U$ was [1]

$$0.15 : \mathtt{cd(Dir, mkdir)} \leftarrow \mathtt{mkdir(Dir, com)} \quad \text{and} \quad 0.08 : \mathtt{cd(\_, mkdir)} \leftarrow \mathtt{mkdir(Dir, com)}$$

favoring changing to the directory just made. This knowledge cannot be captured in $N$

$$0.25 : \quad \mathtt{cd(\_, mkdir)} \quad \leftarrow \quad \mathtt{mkdir(Dir, com)}.$$

The results clearly show that abstraction through variables and unification can be beneficial for some applications, i.e., **(B1)** and **(B2)** hold.

## 6. Real World Applications

Our intentions here are to investigate whether LOHMMs can be applied to real world domains. More precisely, we will investigate whether benefits **(B1)** and **(B2)** can also be exploited in real world application domains. Additionally, we will investigate whether

**(B3)** LOHMMs are competitive with ILP algorithms that can also utilize unification and abstraction through variables, and

**(B4)** LOHMMs can handle tree-structured data similar to PCFGs.

To this aim, we conducted experiments on two bioinformatics application domains: protein fold recognition (Kersting et al., 2003) and mRNA signal structure detection (Horváth et al., 2001). Both application domains are multiclass problems with five different classes each.

1. The sum of probabilities is not the same ($0.15 + 0.08 = 0.23 \neq 0.25$) because of the use of pseudo counts and because of the subliminal non-determinism (w.r.t. abstract states) in $U$, i.e., in case that the first transition fires, the second one also fires.

## 6.1 Methodology

In order to tackle the multiclass problem with LOHMMs, we followed a *plug-in estimate* approach. Let $\{c_1, c_2, \ldots, c_k\}$ be the set of possible classes. Given a finite set of training examples $\{(x_i, y_i)\}_{i=1}^n \subseteq \mathcal{X} \times \{c_1, c_2, \ldots, c_n\}$, one tries to find $f : \mathcal{X} \to \{c_1, c_2, \ldots, c_k\}$

$$f(x) = \arg \max_{c \in \{c_1, c_2, \ldots, c_k\}} P(x \mid M, \lambda_c^*) \cdot P(c) \ . \tag{5}$$

with low approximation error on the training data as well as on unseen examples. In Equation (5), $M$ denotes the model structure which is the same for all classes, $\lambda_c^*$ denotes the maximum likelihood parameters of $M$ for class $c$ estimated on the training examples with $y_i = c$ only, and $P(c)$ is the prior class distribution.

We implemented the Baum-Welch algorithm (with pseudocounts $m$, see line 3) for maximum likelihood parameter estimation using the Prolog system Yap-4.4.4. In all experiments, we set $m = 1$ and let the Baum-Welch algorithm stop if the change in log-likelihood was less than 0.1 from one iteration to the next. The experiments were ran on a Pentium-IV 3.2 GHz Linux machine.

## 6.2 Protein Fold Recognition

Protein fold recognition is concerned with how proteins fold in nature, i.e., their three-dimensional structures. This is an important problem as the biological functions of proteins depend on the way they fold. A common approach is to use database searches to find proteins (of known fold) similar to a newly discovered protein (of unknown fold). To facilitate protein fold recognition, several expert-based classification schemes of proteins have been developed that group the current set of known protein structures according to the similarity of their folds. For instance, the *structural classification of proteins* (Hubbard et al., 1997) (SCOP) database hierarchically organizes proteins according to their structures and evolutionary origin. From a machine learning perspective, SCOP induces a classification problem: given a protein of unknown fold, assign it to the best matching group of the classification scheme. This *protein fold classification* problem has been investigated by Turcotte et al. (2001) based on the inductive logic programming (ILP) system PROGOL and by Kersting et al. (2003) based on LOHMMs.

The secondary structure of protein domains[2] can elegantly be represented as logical sequences. For example, the secondary structure of the Ribosomal protein L4 is represented as

$$\begin{array}{l}\texttt{st}(\texttt{null}, 2), \texttt{he}(\texttt{right}, \texttt{alpha}, 6), \texttt{st}(\texttt{plus}, 2), \texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 2),\\ \texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 3), \texttt{he}(\texttt{right}, \texttt{alpha}, 4), \texttt{st}(\texttt{plus}, 1), \texttt{he}(\texttt{hright}, \texttt{alpha}, 6)\end{array}$$

Helices of a certain type, orientation and length $\texttt{he}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length})$, and strands of a certain orientation and length $\texttt{st}(\mathit{StrandOrientation}, \mathit{Length})$ are atoms over logical predicates. The application of traditional HMMs to such sequences requires one to either ignore the structure of helices and strands, which results in a loss of information, or to take all possible combinations (of arguments such as orientation and length) into account, which leads to a combinatorial explosion in the number of parameters

2. A domain can be viewed as a sub-section of a protein which appears in a number of distantly related proteins and which can fold independently of the rest of the protein.

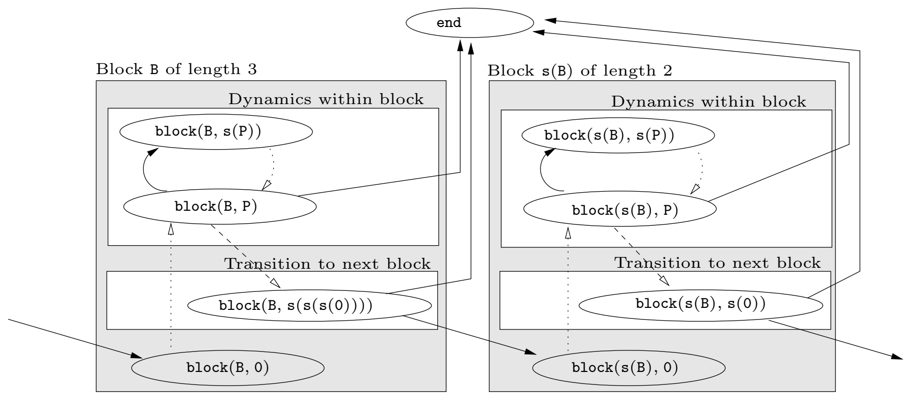
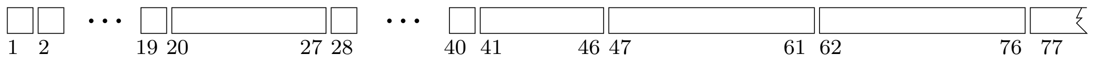


Figure 4: Scheme of a left-to-right LOHMM block model.

The results reported by Kersting et al. (2003) indicate that LOHMMs are well-suited for protein fold classification: the number of parameters of a LOHMM can by an order of magnitude be smaller than the number of a corresponding HMM (120 versus approximately 62000) and the generalization performance, a 74% accuracy, is comparable to Turcotte et al. (2001)'s result based on the ILP system Progol, a 75% accuracy. Kersting et al. (2003), however, do not cross-validate their results nor investigate – as it is common in bioinformatics – the impact of primary sequence similarity on the classification accuracy. For instance, the two most commonly requested ASTRAL subsets are the subset of sequences with less than 95% identity to each other (95 cut) and with less than 40% identity to each other (40 cut). Motivated by this, we conducted the following new experiments.

The data consists of logical sequences of the secondary structure of protein domains. As in (Kersting et al., 2003), the task is to predict one of the five most populated SCOP folds of alpha and beta proteins (a/b): TIM beta/alpha-barrel (fold 1), NAD(P)-binding Rossmann-fold domains (fold 2), Ribosomal protein L4 (fold 23), Cysteine hydrolase (fold 37), and Phosphotyrosine protein phosphatases I-like (fold 55). The class of a/b proteins consists of proteins with mainly parallel beta sheets (beta-alpha-beta units). The data have been extracted automatically from the ASTRAL dataset version 1.65 (Chandonia et al., 2004) for the 95 cut and for the 40 cut. As in (Kersting et al., 2003), we consider strands and helices only, i.e., coils and isolated strands are discarded. For the 95 cut, this yields 816 logical sequences consisting of in total 22210 ground atoms. The number of sequences in the classes are listed as 293, 151, 87, 195, and 90. For the 40 cut, this yields 523 logical sequences consisting of in total 14986 ground atoms. The number of sequences in the classes are listed as 182, 100, 66, 122, and 53.

**LOHMM structure:** The used LOHMM structure follows a left-to-right block topology, see Figure 4, to model blocks of consecutive helices (resp. strands). Being in a *Block* of some size $s$, say 3, the model will remain in the same block for $s = 3$ time steps. A similar idea has been used by Koivisto et al. (2002, 2004) to model haplotypes. In contrast to common HMM block models (Won et al., 2004), the transition parameters are shared within each block and one can ensure that the model makes a transition to the next state s(*Block*) only at the end of a block; in our example after exactly 3 intra-block transitions. Furthermore, there are specific abstract transitions for all helix types and strand orientations to model the priori distribution, the intra- and the inter-block transitions. The number of blocks and their sizes were chosen according to the empirical distribution over sequence lengths in the data so that the beginning and the ending of protein domains was likely captured in detail. This yield the following block structure



where the numbers denote the positions within protein domains. Furthermore, note that the last block gathers all remaining transitions. The blocks themselves are modelled using hidden abstract states over

$$\mathtt{hc}(\mathit{HelixType}, \mathit{HelixOrientation}, \mathit{Length}, \mathit{Block}) \text{ and } \mathtt{sc}(\mathit{StrandOrientation}, \mathit{Length}, \mathit{Block})\ .$$

Here, *Length* denotes the number of consecutive bases the structure element consists of. The length was discretized into 10 bins such that the original lengths were uniformally distributed. In total, the LOHMM has 295 parameters. The corresponding HMM without parameter sharing has more than 65200 parameters. This clearly confirms **(B1)**.

**Results:** We performed a 10-fold cross-validation. On the 95 cut dataset, the accuracy was 76% and took approx. 25 minutes per cross-validation iteration; on the 40 cut, the accuracy was 73% and took approx. 12 minutes per cross-validation iteration. The results validate Kersting et al. (2003)'s results and, in turn, clearly show that **(B3)** holds. Moreover, the novel results on the 40 cut dataset indicate that the similarities detected by the LOHMMs between the protein domain structures were not accompanied by high sequence similarity.

## 6.3 mRNA Signal Structure Detection

mRNA sequences consist of bases (guanine, adenine, uracil, cytosine) and fold intramolecularly to form a number of short base-paired stems (Durbin et al., 1998). This base-paired structure is called the *secondary structure*, cf. Figures 5 and 6. The secondary structure contains special subsequences called signal structures that are responsible for special biological functions, such as RNA-protein interactions and cellular transport. The function of each signal structure class is based on the common characteristic binding site of all class elements. The elements are not necessarily identical but very similar. They can vary in topology (tree structure), in size (number of constituting bases), and in base sequence.

The goal of our experiments was to recognize instances of signal structures classes in mRNA molecules. The first application of relational learning to recognize the signal structure class of mRNA molecules was described in (Bohnebeck et al., 1998; Horváth et al., 2001) where the relational instance-based learner RIBL was applied. The dataset [3] we used was similar to the one described by Horváth et al. (2001). It consisted of 93 mRNA secondary structure sequences. More precisely, it was composed of 15 and 5 SECIS (Selenocysteine Insertion Sequence), 27 IRE (Iron Responsive Element), 36 TAR (Trans Activating Region) and 10 histone stem loops constituting five classes.

3. The dataset is not the same as described in (Horváth et al., 2001) because we could not obtain the original dataset. We will compare to the smaller data set used in (Horváth et al., 2001) which consisted of 66 signal structures and is very close to our data set. On a larger data set (with 400 structures) (Horváth et al., 2001) report an error rate of 3.8% .

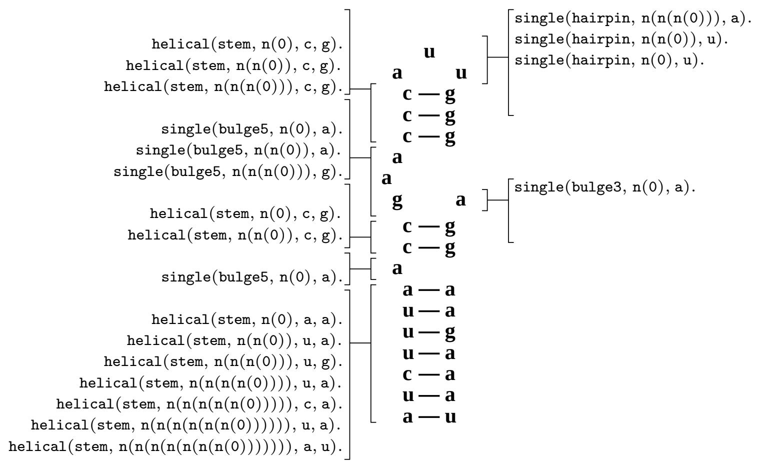

helical(stem, n(0), c, g).
helical(stem, n(n(0)), c, g).
helical(stem, n(n(n(0))), c, g).
single(bulge5, n(0), a).
single(bulge5, n(n(0)), a).
single(bulge5, n(n(n(0))), g).
helical(stem, n(0), c, g).
helical(stem, n(n(0)), c, g).
single(bulge5, n(0), a).
helical(stem, n(0), a, a).
helical(stem, n(n(0)), u, a).
helical(stem, n(n(n(0))), u, g).
helical(stem, n(n(n(n(0)))), u, a).
helical(stem, n(n(n(n(n(0))))), c, a).
helical(stem, n(n(n(n(n(n(0)))))), u, a).
helical(stem, n(n(n(n(n(n(n(0))))))), a, u).
u
a u
c — g
c — g
c — g
a
a
g a
c — g
c — g
a
a — a
u — a
u — g
u — a
c — a
u — a
a — u
single(hairpin, n(n(n(0))), a).
single(hairpin, n(n(0)), u).
single(hairpin, n(0), u).
single(bulge3, n(0), a).

Figure 5: The *chain* representation of a SECIS signal structure. The ground atoms are ordered clockwise starting with `helical(stem, n(n(n(n(n(n(n(0))))))), a, u)` at the lower left-hand side corner.

The secondary structure is composed of different building blocks such as stacking region, hairpin loops, interior loops etc. In contrast to the secondary structure of proteins that forms chains, the secondary structure of mRNA forms a tree. As trees can not easily be handled using HMMs, mRNA secondary structure data is more challenging than that of proteins. Moreover, Horváth et al. (2001) report that making the tree structure available to RIBL as background knowledge had an influence on the classification accuracy. More precisely, using a simple chain representation RIBL achieved a 77.2% leave-one-out cross-validation (LOO) accuracy whereas using the tree structure as background knowledge RIBL achieved a 95.4% LOO accuracy.

We followed Horváth et al.'s experimental setup, that is, we adapted their data representations to LOHMMs and compared a chain model with a tree model.

**Chain Representation:** In the *chain* representation (see also Figure 5), signal structures are described by `single`(*TypeSingle*, *Position*, *Acid*) or `helical`(*TypeHelical*, *Position*, *Acid*, *Acid*). Depending on its type, a structure element is represented by either `single`/3 or `helical`/4. Their first argument *TypeSingle* (resp. *TypeHelical*) specifies the type of the structure element, i.e., `single`, `bulge3`, `bulge5`, `hairpin` (resp. `stem`). The argument *Position* is the position of the sequence element within the corresponding structure element counted down, i.e.[4], $\{\mathtt{n}^{13}(\mathtt{0}), \mathtt{n}^{12}(\mathtt{0}), \ldots, \mathtt{n}^{1}(\mathtt{0})\}$. The maximal position was set to 13 as this was the maximal position observed in the data. The last argument encodes the observed nucleotide (pair).

The used LOHMM structure follows again the left-to-right block structure shown in Figure 4. Its underlying idea is to model blocks of consecutive helical structure elements. The hidden states are modelled using `single`(*TypeSingle*, *Position*, *Acid*, *Block*)

4. $\mathtt{n}^m(\mathtt{0})$ is shorthand for the recursive application of the functor `n` on `0` $m$ times, i.e., for position $m$.

and $\texttt{helical}(\mathit{TypeHelical}, \mathit{Position}, \mathit{Acid}, \mathit{Acid}, \mathit{Block})$. Being in a *Block* of consecutive helical (resp. single) structure elements, the model will remain in the *Block* or transition to a `single` element. The transition to a single (resp. helical) element only occurs at *Position* $\texttt{n(0)}$. At all other positions $\texttt{n}(\mathit{Position})$, there were transitions from helical (resp. single) structure elements to helical (resp. single) structure elements at *Position* capturing the dynamics of the nucleotide pairs (resp. nucleotides) within structure elements. For instance, the transitions for block $\texttt{n(0)}$ at position $\texttt{n(n(0))}$ were

$$\begin{array}{lcl}
a: & \texttt{he(stem,n(0),X,Y,n(0))} \xleftarrow{p_a:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))} \\
b: & \texttt{he(stem,n(0),Y,X,n(0))} \xleftarrow{p_b:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))} \\
c: & \texttt{he(stem,n(0),X,\_,n(0))} \xleftarrow{p_c:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))} \\
d: & \texttt{he(stem,n(0),\_,Y,n(0))} \xleftarrow{p_d:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))} \\
e: & \texttt{he(stem,n(0),\_,\_,n(0))} \xleftarrow{p_e:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))}
\end{array}$$

In total, there were 5 possible blocks as this was the maximal number of blocks of consecutive helical structure elements observed in the data. Overall, the LOHMM has 702 parameters. In contrast, the corresponding HMM has more than 16600 transitions validating **(B1)**.

**Results:** The LOO test log-likelihood was $-63.7$, and an EM iteration took on average 26 seconds.

Without the unification-based transitions *b*-*d*, i.e., using only the abstract transitions

$$\begin{array}{lcl}
a: & \texttt{he(stem,n(0),X,Y,n(0))} \xleftarrow{p_a:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))} \\
e: & \texttt{he(stem,n(0),\_,\_,n(0))} \xleftarrow{p_e:\texttt{he(stem,n(0),X,Y)}} & \texttt{he(stem,n(n(0)),X,Y,n(0)))},
\end{array}$$

the model has 506 parameters. The LOO test log-likelihood was $-64.21$, and an EM iteration took on average 20 seconds. The difference in LOO test log-likelihood is statistically significant (paired $t$-test, $p = 0.01$).

Omitting even transition $a$, the LOO test log-likelihood dropped to $-66.06$, and the average time per EM iteration was 18 seconds. The model has 341 parameters. The difference in average LOO log-likelihood is statistically significant (paired $t$-test, $p = 0.001$).

The results clearly show that unification can yield better LOO test log-likelihoods, i.e., **(B2)** holds.

**Tree Representation:** In the *tree* representation (see Figure 6 **(a)**), the idea is to capture the tree structure formed by the secondary structure elements, see Figure 6 **(b)**. Each training instance is described as a sequence of ground facts over

$$\begin{array}{l}
\texttt{root(0,root,}\#\mathit{Children}), \\
\texttt{helical}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}), \\
\texttt{nucleotide\_pair}(\mathit{BasePair}), \\
\texttt{single}(\mathit{ID}, \mathit{ParentID}, \#\mathit{Children}, \mathit{Type}, \mathit{Size}), \\
\texttt{nucleotide}(\mathit{Base})\ .
\end{array}$$

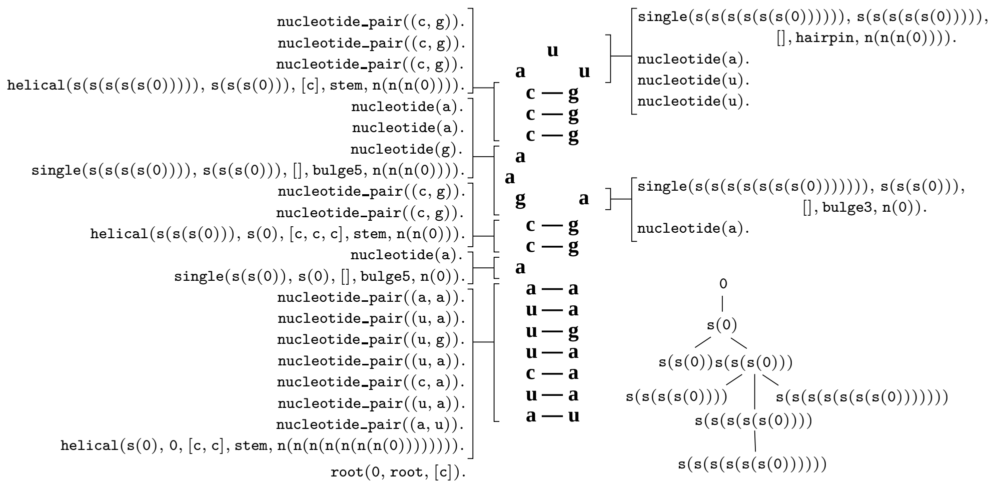


Figure 6: The *tree* representation of a SECIS signal structure. **(a)** The logical sequence, i.e., the sequence of ground atoms representing the SECIS signal structure. The ground atoms are ordered clockwise starting with root(0, root, [c]) in the lower left-hand side corner. **(b)** The tree formed by the secondary structure elements.

Here, *ID* and *ParentID* are natural numbers 0, s(0), s(s(0)), ... encoding the child-parent relation, *#Children* denotes the number[5] of children [], [c], [c, c], ..., *Type* is the type of the structure element such as stem, hairpin, ..., and *Size* is a natural number 0, n(0), n(n(0)), ... Atoms root(0, root, *#Children*) are used to root the topology. The maximal *#Children* was 9 and the maximal *Size* was 13 as this was the maximal value observed in the data.

As trees can not easily be handled using HMMs, we used a LOHMM which basically encodes a PCFG. Due to Theorem 2, this is possible. The used LOHMM structure can be found in Appendix E. It processes the mRNA trees in in-order. Unification is only used for parsing the tree. As for the chain representation, we used a *Position* argument in the hidden states to encode the dynamics of nucleotides (nucleotide pairs) within secondary structure elements. The maximal *Position* was again 13. In contrast to the chain representation, nucleotide pairs such as (a, u) are treated as constants. Thus, the argument *BasePair* consists of 16 elements.

**Results:** The LOO test log-likelihood was $-55.56$. Thus, exploiting the tree structure yields better probabilistic models. On average, an EM iteration took 14 seconds. Overall, the result shows that **(B4)** holds.

Although the Baum-Welch algorithm attempts to maximize a different objective function, namely the likelihood of the data, it is interesting to compare LOHMMs and RIBL in terms of classification accuracy.

5. Here, we use the Prolog short hand notation [·] for lists. A list either is the constant [] representing the empty list, or is a compound term with functor ./2 and two arguments, which are respectively the head and tail of the list. Thus [a, b, c] is the compound term .(a, .(b, .(c, []))).

**Classification Accuracy:** On the *chain* representation, the LOO accuracies of all LOHMMs were 99% (92/93). This is a considerable improvement on RIBL's 77.2% (51/66) LOO accuracy for this representation. On the *tree* representation, the LOHMM also achieved a LOO accuracy of 99% (92/93). This is comparable to RIBL's LOO accuracy of 97% (64/66) on this kind of representation.

Thus, already the chain LOHMMs show marked increases in LOO accuracy when compared to RIBL (Horváth et al., 2001). In order to achieve similar LOO accuracies, Horváth et al. (2001) had to make the tree structure available to RIBL as background knowledge. For LOHMMs, this had a significant influence on the LOO test log-likelihood, but not on the LOO accuracies. This clearly supports **(B3)**. Moreover, according to Horváth et al., the mRNA application can also be considered a success in terms of the application domain, although this was not the primary goal of our experiments. There exist also alternative parameter estimation techniques and other models, such as covariance models (Eddy & Durbin, 1994) or pair hidden Markov models (Sakakibara, 2003), that might have been used as well as a basis for comparison. However, as LOHMMs employ (inductive) logic programming principles, it is appropriate to compare with other systems within this paradigm such as RIBL.

## 7. Related Work

LOHMMs combine two different research directions. On the one hand, they are related to several extensions of HMMs and probabilistic grammars. On the other hand, they are also related to the recent interest in combining inductive logic programming principles with probability theory (De Raedt & Kersting, 2003, 2004).

In the first type of approaches, the underlying idea is to *upgrade* HMMs and probabilistic grammars to represent more structured state spaces.

Hierarchical HMMs (Fine et al., 1998), factorial HMMs (Ghahramani & Jordan, 1997), and HMMs based on tree automata (Frasconi et al., 2002) decompose the state variables into smaller units. 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, 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. As our experimental evidence shows, sharing information among abstract states by means of unification can lead to more accurate model estimation. The same holds for *relational Markov models* (RMMs) (Anderson et al., 2002) to which LOHMMs are most closely related. In RMMs, 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 between LOHMMs and RMMs are that RMMs do not either support variable binding nor unification nor hidden states.

The equivalent of HMMs for context-free languages are *probabilistic context-free grammars* (PCFGs). Like HMMs, they do not consider sequences of logical atoms and do not employ unification. Nevertheless, there is a formal resemblance between the Baum-Welch algorithms for LOHMMs and for PCFGs. In case that a LOHMM encodes a PCFG both algorithms are identical from a theoretical point of view. They re-estimate the parameters as the ratio of the expected number of times a transition (resp. production) is used and the expected number of times a transition (resp. production) might have been used. The proof of Theorem 2 assumes that the PCFG is given in Greibach normal form[6] (GNF) and uses a pushdown automaton to parse sentences. For grammars in GNF, pushdown automata are common for parsing. In contrast, the actual computations of the Baum-Welch algorithm for PCFGs, the so called Inside-Outside algorithm (Baker, 1979; Lari & Young, 1990), is usually formulated for grammars in *Chomsky normal form*[7]. The Inside-Outside algorithm can make use of the efficient CYK algorithm (Hopcroft & Ullman, 1979) for parsing strings.

An alternative to learning PCFGs from strings only is to learn from more structured data such as *skeletons*, which are derivation trees with the nonterminal nodes removed (Levy & Joshi, 1978). Skeletons are exactly the set of trees accepted by *skeletal tree automata* (STA). Informally, an STA, when given a tree as input, processes the tree bottom up, assigning a state to each node based on the states of that node's children. The STA accepts a tree iff it assigns a final state to the root of the tree. Due to this automata-based characterization of the skeletons of derivation trees, the learning problem of (P)CFGs can be reduced to the problem of an STA. In particular, STA techniques have been adapted to learning tree grammars and (P)CFGs (Sakakibara, 1992; Sakakibara et al., 1994) efficiently.

PCFGs have been extended in several ways. Most closely related to LOHMMs are *unification-based grammars* which have been extensively studied in computational linguistics. Examples are (stochastic) attribute-value grammars (Abney, 1997), probabilistic feature grammars (Goodman, 1997), head-driven phrase structure grammars (Pollard & Sag, 1994), and lexical-functional grammars (Bresnan, 2001). For learning within such frameworks, methods from undirected graphical models are used; see (Johnson, 2003) for a description of some recent work. The key difference to LOHMMs is that only nonterminals are replaced with structured, more complex entities. Thus, observation sequences of flat symbols and not of atoms are modelled. Goodman's *probabilistic feature grammars* are an exception. They treat terminals and nonterminals as vectors of features. No abstraction is made, i.e., the feature vectors are ground instances, and no unification can be employed. Therefore, the number of parameters that needs to be estimated becomes easily very large, data sparsity is a serious problem. Goodman applied smoothing to overcome the problem.

LOHMMs are generally related to (stochastic) tree automata, see e.g. (Carrasco, Oncina, & Calera-Rubio, 2001). Reconsider the Unix command sequence $\mathtt{mkdir(vt100x)}, \mathtt{mv(new*, vt100x)}, \mathtt{ls(vt100x)}, \mathtt{cd(vt100x)}$ . Each atom forms a tree, see Figure 7 **(a)**, and, indeed, the whole sequence of atoms also forms a (degenerated) tree, see Figure 7 **(b)**. Tree automata process single trees vertically, e.g., bottom-up. A state in the automaton is assigned to every node in the tree. The state depends on the node label and on the states associated to the siblings of the node. They do not focus on sequential domains. In contrast, LOHMMs are intended for learning in sequential domains. They process sequences of trees horizontally, i.e., from left to right. Furthermore, unification

6. A grammar is in GNF iff all productions are of the form $\mathtt{A} \leftarrow \mathtt{a}V$ where $\mathtt{A}$ is a variable, $\mathtt{a}$ is exactly one terminal and $V$ is a string of none or more variables.
7. A grammar is in CNF iff every production is of the form $\mathtt{A} \leftarrow \mathtt{B}, \mathtt{C}$ or $\mathtt{A} \leftarrow \mathtt{a}$ where $\mathtt{A}, \mathtt{B}$ and $\mathtt{C}$ are variables, and $\mathtt{a}$ is a terminal.

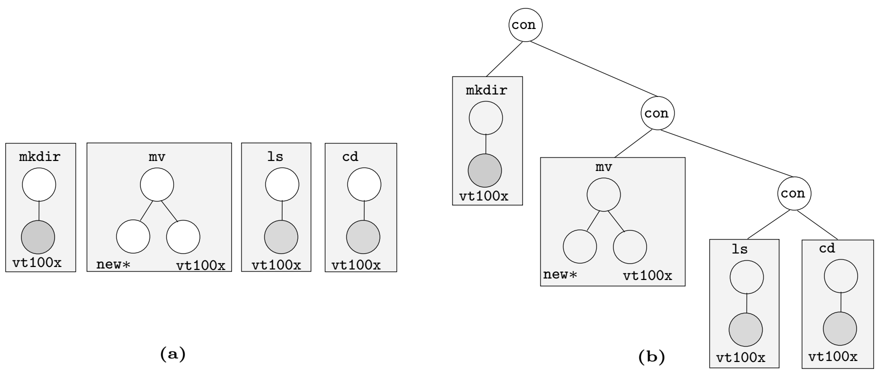


Figure 7: **(a)** Each atom in the logical sequence mkdir(vt100x), mv(new∗, vt100x), ls(vt100x), cd(vt100x) forms a tree. The shaded nodes denote shared labels among the trees. **(b)** The same sequence represented as a single tree. The predicate con/2 represents the concatenation operator.

is used to share information between consecutive sequence elements. As Figure 7 **(b)** illustrates, tree automata can only employ this information when allowing higher-order transitions, i.e., states depend on their node labels and on the states associated to predecessors $1, 2, \ldots$ levels down the tree.

In the second type of approaches, most attention has been devoted to developing highly expressive formalisms, such as e.g. PCUP (Eisele, 1994), PCLP (Riezler, 1998), SLPs (Muggleton, 1996), PLPs (Ngo & Haddawy, 1997), RBNs (Jaeger, 1997), PRMs (Friedman et al., 1999), PRISM (Sato & Kameya, 2001), BLPs (Kersting & De Raedt, 2001b, 2001a), and DPRMs (Sanghai et al., 2003). LOHMMs can be seen as an attempt towards *downgrading* such highly expressive frameworks. Indeed, applying the main idea underlying LOHMMs to non-regular probabilistic grammar, i.e., replacing flat symbols with atoms, yields – in principle – stochastic logic programs (Muggleton, 1996). 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* (Muggleton & De Raedt, 1994) and multi-relational data mining (Džeroski & Lavrač, 2001).

## 8. Conclusions

Logical hidden Markov models, a new formalism for representing probability distributions over sequences of logical atoms, have been introduced and solutions to the three central inference problems (evaluation, most likely state sequence and parameter estimation) have been provided. Experiments have demonstrated that unification can improve generalization accuracy, that the number of parameters of a LOHMM can be an order of magnitude smaller than the number of parameters of the corresponding HMM, that the solutions presented perform well in practice and also that LOHMMs possess several advantages over traditional HMMs for applications involving structured sequences.

**Acknowledgments** The authors thank Andreas Karwath and Johannes Horstmann for interesting collaborations on the protein data; Ingo Thon for interesting collaboration on analyzing the Unix command sequences; and Saul Greenberg for providing the Unix command sequence data. The authors would also like to thank the anonymous reviewers for comments which considerably improved the paper. This research was partly supported by the European Union IST programme under contract numbers IST-2001-33053 and FP6-508861 (Application of Probabilistic Inductive Logic Programming (APrIL) I and II). Tapani Raiko was supported by a Marie Curie fellowship at DAISY, HPMT-CT-2001-00251.

## References

Abney, S. (1997). Stochastic Attribute-Value Grammars. *Computational Linguistics*, *23*(4), 597–618.

Abney, S., McAllester, D., & Pereira, F. (1999). Relating probabilistic grammars and automata. In *Proceedings of 37th Annual Meeting of the Association for Computational Linguistics (ACL-1999)*, pp. 542–549. Morgan Kaufmann.

Anderson, C., Domingos, P., & Weld, D. (2002). Relational Markov Models and their Application to Adaptive Web Navigation. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining (KDD-2002)*, pp. 143–152 Edmonton, Canada. ACM Press.

Baker, J. (1979). Trainable grammars for speech recognition. In *Speech communication paper presented at th 97th Meeting of the Acoustical Society of America*, pp. 547–550 Boston, MA.

Bauer, H. (1991). *Wahrscheinlichkeitstheorie* (4. edition). Walter de Gruyter, Berlin, New York.

Baum, L. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, *3*, 1–8.

Bohnebeck, U., Horváth, T., & Wrobel, S. (1998). Term comparison in first-order similarity measures. In *Proceedings of the Eigth International Conference on Inductive Logic Programming (ILP-98)*, Vol. 1446 of *LNCS*, pp. 65–79. Springer.

Bresnan, J. (2001). *Lexical-Functional Syntax.* Blackwell, Malden, MA.

Carrasco, R., Oncina, J., & Calera-Rubio, J. (2001). Stochastic inference of regular tree languages. *Machine Learning*, *44*(1/2), 185–197.

Chandonia, J., Hon, G., Walker, N., Lo Conte, L., P.Koehl, & Brenner, S. (2004). The ASTRAL compendium in 2004. *Nucleic Acids Research*, *32*, D189–D192.

Davison, B., & Hirsh, H. (1998). Predicting Sequences of User Actions. In *Predicting the Future: AI Approaches to Time-Series Analysis*, pp. 5–12. AAAI Press.

De Raedt, L., & Kersting, K. (2003). Probabilistic Logic Learning. *ACM-SIGKDD Explorations: Special issue on Multi-Relational Data Mining*, *5*(1), 31–48.

De Raedt, L., & Kersting, K. (2004). Probabilistic Inductive Logic Programming. In Ben-David, S., Case, J., & Maruoka, A. (Eds.), *Proceedings of the 15th International Conference on Algorithmic Learning Theory (ALT-2004)*, Vol. 3244 of *LNCS*, pp. 19–36 Padova, Italy. Springer.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press.

Džeroski, S., & Lavrač, N. (Eds.). (2001). *Relational data mining.* Springer-Verlag, Berlin.

Eddy, S., & Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucleic Acids Res.*, *22*(11), 2079–2088.

Eisele, A. (1994). Towards probabilistic extensions of contraint-based grammars. In Dörne, J. (Ed.), *Computational Aspects of Constraint-Based Linguistics Decription-II.* DYNA-2 deliverable R1.2.B.

Fine, S., Singer, Y., & Tishby, N. (1998). The hierarchical hidden markov model: analysis and applications. *Machine Learning*, *32*, 41–62.

Frasconi, P., Soda, G., & Vullo, A. (2002). Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, *18*, 195–217.

Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-1999)*, pp. 1300–1307. Morgan Kaufmann.

Fristedt, B., & Gray, L. (1997). *A Modern Approach to Probability Theory.* Probability and its applications. Birkhäuser Boston.

Ghahramani, Z., & Jordan, M. (1997). Factorial hidden Markov models. *Machine Learning*, *29*, 245–273.

Goodman, J. (1997). Probabilistic feature grammars. In *Proceedings of the Fifth International Workshop on Parsing Technologies (IWPT-97)* Boston, MA, USA.

Greenberg, S. (1988). Using Unix: collected traces of 168 users. Tech. rep., Dept. of Computer Science, University of Calgary, Alberta.

Hopcroft, J., & Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company.

Horváth, T., Wrobel, S., & Bohnebeck, U. (2001). Relational Instance-Based learning with Lists and Terms. *Machine Learning*, *43*(1/2), 53–80.

Hubbard, T., Murzin, A., Brenner, S., & Chotia, C. (1997). *SCOP*: a structural classification of proteins database. *NAR*, *27*(1), 236–239.

Jacobs, N., & Blockeel, H. (2001). The Learning Shell: Automated Macro Construction. In *User Modeling 2001*, pp. 34–43.

Jaeger, M. (1997). Relational Bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 266–273. Morgan Kaufmann.

Katz, S. (1987). Estimation of probabilities from sparse data for hte language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing (ASSP)*, *35*, 400–401.

Kersting, K., & De Raedt, L. (2001a). Adaptive Bayesian Logic Programs. In Rouveirol, C., & Sebag, M. (Eds.), *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, Vol. 2157 of *LNAI*, pp. 118–131. Springer.

Kersting, K., & De Raedt, L. (2001b). Towards Combining Inductive Logic Programming with Bayesian Networks. In Rouveirol, C., & Sebag, M. (Eds.), *Proceedings of the 11th International Conference on Inductive Logic Programming (ILP-01)*, Vol. 2157 of *LNAI*, pp. 118–131. Springer.

Kersting, K., & Raiko, T. (2005). 'Say EM' for Selecting Probabilistic Models for Logical Sequences. In Bacchus, F., & Jaakkola, T. (Eds.), *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, UAI 2005*, pp. 300–307 Edinburgh, Scotland.

Kersting, K., Raiko, T., Kramer, S., & De Raedt, L. (2003). Towards discovering structural signatures of protein folds based on logical hidden markov models. In Altman, R., Dunker, A., Hunter, L., Jung, T., & Klein, T. (Eds.), *Proceedings of the Pacific Symposium on Biocomputing (PSB-03)*, pp. 192–203 Kauai, Hawaii, USA. World Scientific.

Koivisto, M., Kivioja, T., Mannila, H., Rastas, P., & Ukkonen, E. (2004). Hidden Markov Modelling Techniques for Haplotype Analysis. In Ben-David, S., Case, J., & Maruoka, A. (Eds.), *Proceedings of 15th International Conference on Algorithmic Learning Theory (ALT-04)*, Vol. 3244 of *LNCS*, pp. 37–52. Springer.

Koivisto, M., Perola, M., Varilo, T., Hennah, W., Ekelund, J., Lukk, M., Peltonen, L., Ukkonen, E., & Mannila, H. (2002). An MDL method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In Altman, R., Dunker, A., Hunter, L., Jung, T., & Klein, T. (Eds.), *Proceedings of the Pacific Symposium on Biocomputing (PSB-02)*, pp. 502–513. World Scientific.

Korvemaker, B., & Greiner, R. (2000). Predicting UNIX command files: Adjusting to user patterns. In *Adaptive User Interfaces: Papers from the 2000 AAAI Spring Symposium*, pp. 59–64.

Kulp, D., Haussler, D., Reese, M., & Eeckman, F. (1996). A Generalized Hidden Markov Model for the Recognition of Human Genes in DNA. In States, D., Agarwal, P., Gaasterland, T., Hunter, L., & Smith, R. (Eds.), *Proceedings of the Fourth International Conference on Intelligent Systems for Molecular Biology,(ISMB-96)*, pp. 134–142 St. Louis, MO, USA. AAAI.

Lane, T. (1999). Hidden Markov Models for Human/Computer Interface Modeling. In Rudström, Å. (Ed.), *Proceedings of the IJCAI-99 Workshop on Learning about Users*, pp. 35–44 Stockholm, Sweden.

Lari, K., & Young, S. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

Levy, L., & Joshi, A. (1978). Skeletal structural descriptions. *Information and Control*, *2*(2), 192–211.

McLachlan, G., & Krishnan, T. (1997). *The EM Algorithm and Extensions.* Wiley, New York.

Mitchell, T. M. (1997). *Machine Learning.* The McGraw-Hill Companies, Inc.

Muggleton, S. (1996). Stochastic logic programs. In De Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, pp. 254–264. IOS Press.

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19*(20), 629–679.

Ngo, L., & Haddawy, P. (1997). Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, *171*, 147–177.

Pollard, C., & Sag, I. (1994). *Head-driven Phrase Structure Grammar.* The University of Chicago Press, Chicago.

Rabiner, L., & Juang, B. (1986). An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, *3*(1), 4–16.

Riezler, S. (1998). Statistical inference and probabilistic modelling for constraint-based nlp. In Schrder, B., Lenders, W., & und T. Portele, W. H. (Eds.), *Proceedings of the 4th Conference on Natural Language Processing (KONVENS-98).* Also as CoRR cs.CL/9905010.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, *97*(1), 23–60.

Sakakibara, Y. (2003). Pair hidden markov models on tree structures. *Bioinformatics*, *19*(Suppl.1), i232–i240.

Sakakibara, Y., Brown, M., Hughey, R., Mian, I., Sjolander, K., & Underwood, R. (1994). Stochastic context-free grammars for tRNA modelling. *Nucleic Acids Research*, *22*(23), 5112–5120.

Sanghai, S., Domingos, P., & Weld, D. (2003). Dynamic probabilistic relational models. In Gottlob, G., & Walsh, T. (Eds.), *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 992–997 Acapulco, Mexico. Morgan Kaufmann.

Sato, T., & Kameya, Y. (2001). Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research (JAIR)*, *15*, 391–454.

Schölkopf, B., & Warmuth, M. (Eds.). (2003). *Learning and Parsing Stochastic Unification-Based Grammars*, Vol. 2777 of *LNCS*. Springer.

Turcotte, M., Muggleton, S., & Sternberg, M. (2001). The effect of relational background knowledge on learning of protein three-dimensional fold signatures. *Machine Learning*, *43*(1/2), 81–95.

Won, K., Prügel-Bennett, A., & Krogh, A. (2004). The Block Hidden Markov Model for Biological Sequence Analysis. In Negoita, M., Howlett, R., & Jain, L. (Eds.), *Proceedings of the Eighth International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES-04)*, Vol. 3213 of *LNCS*, pp. 64–70. Springer.

## Appendix A. Proof of Theorem 1

Let $M = (\Sigma, \mu, \Delta, \Upsilon)$ be a LOHMM. To show that $M$ specifies a time discrete stochastic process, i.e., a sequence of random variables $\langle X_t \rangle_{t=1,2,\ldots}$, where the domains of the random variable $X_t$ is hb($\Sigma$), the Herbrand base over $\Sigma$, we define the *immediate state operator* $T_M$-operator and the *current emission operator* $E_M$-operator.

**Definition 4** *($T_M$-Operator, $E_M$-Operator ) The operators $T_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ and $E_M : 2^{\mathrm{hb}_\Sigma} \to 2^{\mathrm{hb}_\Sigma}$ are*

$$T_M(I) = \{H\sigma_B\sigma_H \mid \exists (p : H \overset{O}{\longleftarrow} B) \in M : B\sigma_B \in I, H\sigma_B\sigma_H \in G_\Sigma(H)\}$$

$$E_M(I) = \{O\sigma_B\sigma_H\sigma_O \mid \exists (p : H \overset{O}{\longleftarrow} B) \in M : B\sigma_B \in I,\ H\sigma_B\sigma_G \in G_\Sigma(H) \text{ and } O\sigma_B\sigma_H\sigma_O \in G_\Sigma(O)\}$$

For each $i = 1, 2, 3, \ldots$, the set $T_M^{i+1}(\{\texttt{start}\}) := T_M(T_M^i(\{\texttt{start}\}))$ with $T_M^1(\{\texttt{start}\}) := T_M(\{\texttt{start}\})$ specifies the state set at clock $i$ which forms a random variable $Y_i$. The set $U_M^i(\{\texttt{start}\})$ specifies the possible symbols emitted when transitioning from $i$ to $i+1$. It forms the variable $U_i$. Each $Y_i$ (resp. $U_i$) can be extended to a random variable $Z_i$ (resp. $U_i$) over hb$_\Sigma$:

$$P(Z_i = z) = \begin{cases} 0.0 & : \ z \notin T_M^i(\{\texttt{start}\}) \\ P(Y_i = z) & : \ \text{otherwise} \end{cases}$$

Figure 8 depicts the influence relation among $Z_i$ and $U_i$. Using standard arguments from probability theory and noting that

$$P(U_i = U_i \mid Z_{i+1} = z_{i+1}, Z_i = z_i) = \frac{P(Z_{i+1} = z_{i+1}, U_i = u_i \mid Z_i)}{\sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)}$$

$$\text{and } P(Z_{i+1} \mid Z_i) = \sum_{u_i} P(Z_{i+1}, u_i \mid Z_i)$$

where the probability distributions are due to equation (1), it is easy to show that Kolmogorov's extension theorem (see (Bauer, 1991; Fristedt & Gray, 1997)) holds. Thus, $M$ specifies a unique probability distribution over $\bigotimes_{i=1}^t (Z_i \times U_i)$ for each $t > 0$ and in the limit $t \to \infty$. $\square$

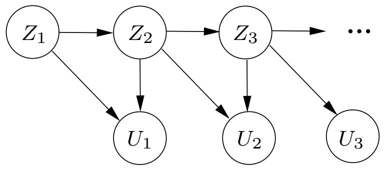


Figure 8: Discrete time stochastic process induced by a LOHMM. The nodes $Z_i$ and $U_i$ represent random variables over $\mathrm{hb}_\Sigma$.

## Appendix B. Moore Representations of LOHMMs

For HMMs, *Moore* representations, i.e., output symbols are associated with states and *Mealy* representations, i.e., output symbols are associated with transitions, are equivalent. In this appendix, we will investigate to which extend this also holds for LOHMMs.

Let $L$ be a Mealy-LOHMM according to definition 3. In the following, we will derive the notation of an equivalent LOHMM $L'$ in Moore representation where there are abstract transitions and abstract emissions (see below). Each predicate $\mathtt{b}/n$ in $L$ is extended to $\mathtt{b}/n+1$ in $L'$. The domains of the first $n$ arguments are the same as for $\mathtt{b}/n$. The last argument will store the observation to be emitted. More precisely, for each abstract transition

$$p : \mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l}) \xleftarrow{\mathtt{o}(\mathtt{v_1},\ldots,\mathtt{v_k})} \mathtt{b}(\mathtt{u_1},\ldots,\mathtt{u_n})$$

in $L$, there is an abstract transition

$$p : \mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l},\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k})) \leftarrow \mathtt{b}(\mathtt{u_1},\ldots,\mathtt{u_n},\_)$$

in $L'$. The primes in $\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k})$ denote that we replaced each free [8] variables $\mathtt{o}(\mathtt{v_1},\ldots,\mathtt{v_k})$ by some distinguished constant symbol, say $\#$. Due to this, it holds that

$$\mu(\mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l})) = \mu(\mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l},\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k}))) \;, \tag{6}$$

and $L'$'s output distribution can be specified using *abstract emissions* which are expressions of the form

$$1.0 : \mathtt{o}(\mathtt{v_1},\ldots,\mathtt{v_k}) \leftarrow \mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l},\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k})) \;. \tag{7}$$

The semantics of an abstract transition in $L'$ is that being in some state $\mathtt{S}'_t \in G_{\Sigma'}(\mathtt{b}(\mathtt{u_1},\ldots,\mathtt{u_n},\_))$ the system will make a transition into state $\mathtt{S}'_{t+1} \in G_{\Sigma'}(\mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l},\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k})))$ with probability

$$p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l},\mathtt{o}(\mathtt{v'_1},\ldots,\mathtt{v'_k})) \mid \sigma_{\mathtt{S}'_t}) \tag{8}$$

where $\sigma_{\mathtt{S}'_t} = \mathrm{mgu}(\mathtt{S}'_t, \mathtt{b}(\mathtt{u_1},\ldots,\mathtt{u_n},\_))$. Due to Equation (6), Equation (8) can be rewritten as

$$p \cdot \mu(\mathtt{S}'_{t+1} \mid \mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l}) \mid \sigma_{\mathtt{S}'_t}) \;.$$

8. A variable $\mathtt{X} \in \mathrm{vars}(\mathtt{o}(\mathtt{v_1},\ldots,\mathtt{v_k}))$ is free iff $\mathtt{X} \notin \mathrm{vars}(\mathtt{h}(\mathtt{w_1},\ldots,\mathtt{w_l})) \cup \mathrm{vars}(\mathtt{b}(\mathtt{u_1},\ldots,\mathtt{u_n}))$.

Due to equation (7), the system will emit the output symbol $\mathtt{o}_{t+1} \in G_{\Sigma'}(\mathtt{o}(\mathtt{v}_1, \ldots, \mathtt{v}_\mathtt{k}))$ in state $\mathtt{S}'_{t+1}$ with probability

$$\mu(\mathtt{o}_{t+1} \mid \mathtt{o}(\mathtt{v}_1, \ldots, \mathtt{v}_\mathtt{k})\sigma_{\mathtt{S}'_{t+1}}\sigma_{\mathtt{S}'_t})$$

where $\sigma_{\mathtt{S}'_{t+1}} = \mathrm{mgu}(\mathtt{h}(\mathtt{w}_1, \ldots, \mathtt{w}_\mathtt{l}, \mathtt{o}(\mathtt{v}'_1, \ldots, \mathtt{v}'_\mathtt{k})), \mathtt{S}'_{t+1})$. Due to the construction of $L'$, there exists a triple $(\mathtt{S}_t, \mathtt{S}_{t+1}, \mathtt{O}_{\mathtt{t+1}})$ in $L$ for each triple $(\mathtt{S}'_t, \mathtt{S}'_{t+1}, \mathtt{O}_{t+1})$, $t > 0$, in $L'$ (and vise versa). Hence,both LOHMMs assign the same overall transition probability.

$L$ and $L'$ differ only in the way the initialize sequences $\langle(\mathtt{S}'_t, \mathtt{S}'_{t+1}, \mathtt{O}_{t+1}\rangle_{t=0,2\ldots,T}$ (resp. $\langle(\mathtt{S}_t, \mathtt{S}_{t+1}, \mathtt{O}_{\mathtt{t+1}}\rangle_{t=0,2\ldots,T})$. Whereas $L$ starts in some state $\mathtt{S}_0$ and makes a transition to $\mathtt{S}_1$ emitting $\mathtt{O}_1$, the Moore-LOHMM $L'$ is supposed to emit a symbol $\mathtt{O}_0$ in $\mathtt{S}'_0$ before making a transition to $\mathtt{S}'_1$. We compensate for this using the prior distribution. The existence of the correct prior distribution for $L'$ can be seen as follows. In $L$, there are only finitely many states reachable at time $t = 1$, i.e, $P_L(q_0 = \mathtt{S}) > 0$ holds for only a finite set of ground states $\mathtt{S}$. The probability $P_L(q_0 = \mathtt{s})$ can be computed similar to $\alpha_1(\mathtt{S})$. We set $t = 1$ in line 6, neglecting the condition on $\mathtt{O}_{t-1}$ in line 10, and dropping $\mu(\mathtt{O}_{t-1} \mid \mathtt{O}\sigma_\mathtt{B}\sigma_\mathtt{H})$ from line 14. Completely listing all states $\mathtt{S} \in S_1$ together with $P_L(q_0 = \mathtt{S})$, i.e., $P_L(q_0 = \mathtt{S}) : \mathtt{S} \leftarrow \mathtt{start}$ , constitutes the prior distribution of $L'$.

The argumentation basically followed the approach to transform a Mealy machine into a Moore machine, see e.g. (Hopcroft & Ullman, 1979). Furthermore, the mapping of a Moore-LOHMM – as introduced in the present section – into a Mealy-LOHMM is straightforward.

## Appendix C. Proof of Theorem 2

Let $T$ be a terminal alphabet and $N$ a nonterminal alphabet. A *probabilistic context-free grammar* (PCFG) $G$ consists of a distinguished start symbol $S \in N$ plus a finite set of productions of the form $p : X \rightarrow \alpha$, where $X \in N$, $\alpha \in (N \cup T)^*$ and $p \in [0, 1]$. For all $X \in N$, $\sum_{:X\rightarrow\alpha} p = 1$. A PCFG defines a stochastic process with sentential forms as states, and leftmost rewriting steps as transitions. We denote a single rewriting operation of the grammar by a single arrow $\rightarrow$. If as a result of one ore more rewriting operations we are able to rewrite $\beta \in (N \cup T)^*$ as a sequence $\gamma \in (N \cup T)^*$ of nonterminals and terminals, then we write $\beta \Rightarrow^* \gamma$. The probability of this rewriting is the product of all probability values associated to productions used in the derivation. We assume $G$ to be consistent, i.e., that the sum of all probabilities of derivations $S \Rightarrow^* \beta$ such that $\beta \in T^*$ sum to 1.0.

We can assume that the PCFG $G$ is in Greibach normal form. This follows from Abney et al. (1999)'s Theorem 6 because $G$ is consistent. Thus, every production $P \in G$ is of the form $p : X \rightarrow aY_1 \ldots Y_n$ for some $n \geq 0$. In order to encode $G$ as a LOHMM $M$, we introduce (1) for each non-terminal symbol $X$ in $G$ a constant symbol $\mathtt{nX}$ and (2) for each terminal symbol $t$ in $G$ a constant symbol $\mathtt{t}$. For each production $P \in G$, we include an abstract transition of the form $p : \mathtt{stack}([\mathtt{nY}_1, \ldots, \mathtt{nY}_\mathtt{n}|\mathtt{S}]) \xleftarrow{\mathtt{a}} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n > 0$, and $p : \mathtt{stack}(\mathtt{S}) \xleftarrow{\mathtt{a}} \mathtt{stack}([\mathtt{nX}|\mathtt{S}])$, if $n = 0$. Furthermore, we include $1.0 : \mathtt{stack}([\mathtt{s}]) \leftarrow \mathtt{start}$ and $1.0 : \mathtt{end} \xleftarrow{\mathtt{end}} \mathtt{stack}([])$. It is now straightforward to prove by induction that $M$ and $G$ are equivalent. $\square$

## Appendix D. Logical Hidden Markov Model for Unix Command Sequences

The LOHMMs described below model Unix command sequences triggered by `mkdir`. To this aim, we transformed the original Greenberg data into a sequence of logical atoms over `com`, `mkdir(Dir, LastCom)`, `ls(Dir, LastCom)`, `cd(Dir, Dir, LastCom)`, `cp(Dir, Dir, LastCom)` and `mv(Dir, Dir, LastCom)`. The domain of `LastCom` was {`start`, `com`, `mkdir`, `ls`, `cd`, `cp`, `mv`}. The domain of `Dir` consisted of all argument entries for `mkdir`, `ls`, `cd`, `cp`, `mv` in the original dataset. Switches, pipes, etc. were neglected, and paths were made absolute. This yields 212 constants in the domain of `Dir`. All original commands, which were not `mkdir`, `ls`, `cd`, `cp`, or `mv`, were represented as `com`. If `mkdir` did not appear within 10 time steps before a command $C \in \{\texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}\}$, $C$ was represented as `com`. Overall, this yields more than 451000 ground states that have to be covered by a Markov model.

The "unification" LOHMM $U$ basically implements a second order Markov model, i.e., the probability of making a transition depends upon the current state and the previous state. It has 542 parameters and the following structure:

$$\begin{array}{rcl rcl}
\texttt{com} & \leftarrow & \texttt{start}. & \texttt{com} & \leftarrow & \texttt{com}. \\
\texttt{mkdir(Dir, start)} & \leftarrow & \texttt{start}. & \texttt{mkdir(Dir, com)} & \leftarrow & \texttt{com}. \\
 & & & \texttt{end} & \leftarrow & \texttt{com}.
\end{array}$$

Furthermore, for each $C \in \{\texttt{start}, \texttt{com}\}$ there are

$$\begin{array}{rcl rcl}
\texttt{mkdir(Dir, com)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{cd(\_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{mkdir(\_, com)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{cp(\_, Dir, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{com} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{cp(Dir, \_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{end} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{cp(\_, \_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{ls(Dir, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{mv(\_, Dir, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{ls(\_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{mv(Dir, \_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. \\
\texttt{cd(Dir, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}. & \texttt{mv(\_, \_, mkdir)} & \leftarrow & \texttt{mkdir(Dir,}C\texttt{)}.
\end{array}$$

together with for each $C \in \{\texttt{mkdir}, \texttt{ls}, \texttt{cd}, \texttt{cp}, \texttt{mv}\}$ and for each $C_1 \in \{\texttt{cd}, \texttt{ls}\}$ (resp. $C_2 \in \{\texttt{cp}, \texttt{mv}\}$)

$$\begin{array}{rcl rcl}
\texttt{mkdir(Dir, com)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{mkdir(\_, com)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{mkdir(\_, com)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{com} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{com} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{end} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{end} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{ls(From,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{ls(Dir,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{ls(To,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{ls(\_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{ls(\_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{cd(Dir,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cd(From,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{cd(\_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cd(To,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{cp(\_, Dir,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cd(\_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{cp(Dir, \_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cp(From, \_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{cp(\_, \_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cp(\_, To,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{mv(\_, Dir,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{cp(\_, \_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{mv(Dir, \_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{mv(From, \_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
\texttt{mv(\_, \_,}C_1\texttt{)} & \leftarrow & C_1\texttt{(Dir,}C\texttt{)}. & \texttt{mv(\_, To,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}. \\
 & & & \texttt{mv(\_, \_,}C_2\texttt{)} & \leftarrow & C_2\texttt{(From, To,}C\texttt{)}.
\end{array}$$

Because all states are fully observable, we omitted the output symbols associated with clauses, and, for the sake of simplicity, we omitted associated probability values.

The "no unification" LOHMM $N$ is the variant of $U$ where no variables were shared such as

$$\begin{array}{rclrcl}
\texttt{mkdir(\_,com)} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. & \texttt{ls(\_,cp)} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. \\
\texttt{com} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. & \texttt{cd(\_,cp)} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. \\
\texttt{end} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. & \texttt{cp(\_,\_,cp)} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}. \\
 & & & \texttt{mv(\_,\_,cp)} & \leftarrow & \texttt{cp(From,To,}C\texttt{)}.
\end{array}$$

Because only transitions are affected, $N$ has 164 parameters less than $U$, i.e., 378.

## Appendix E. Tree-based LOHMM for mRNA Sequences

The LOHMM processes the nodes of mRNA trees in in-order. The structure of the LOHMM is shown at the end of the section. There are copies of the shaded parts. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.

The domain of *#Children* covers the maximal branching factor found in the data, i.e., $\{[\texttt{c}], [\texttt{c},\texttt{c}], \ldots, [\texttt{c},\texttt{c},\texttt{c},\texttt{c},\texttt{c},\texttt{c},\texttt{c},\texttt{c},\texttt{c}]\}$; the domain of *Type* consists of all types occurring in the data, i.e., $\{\texttt{stem}, \texttt{single}, \texttt{bulge3}, \texttt{bulge5}, \texttt{hairpin}\}$; and for *Size*, the domain covers the maximal length of a secondary structure element in the data, i.e., the longest sequence of consecutive bases respectively base pairs constituting a secondary structure element. The length was encoded as $\{\texttt{n}^1(\texttt{0}), \texttt{n}^2(\texttt{0}), \ldots, \texttt{n}^{13}(\texttt{0})\}$ where $\texttt{n}^m(\texttt{0})$ denotes the recursive application of the functor `n` $m$ times. For *Base* and *BasePair*, the domains were the 4 bases respectively the 16 base pairs. In total, there are 491 parameters.

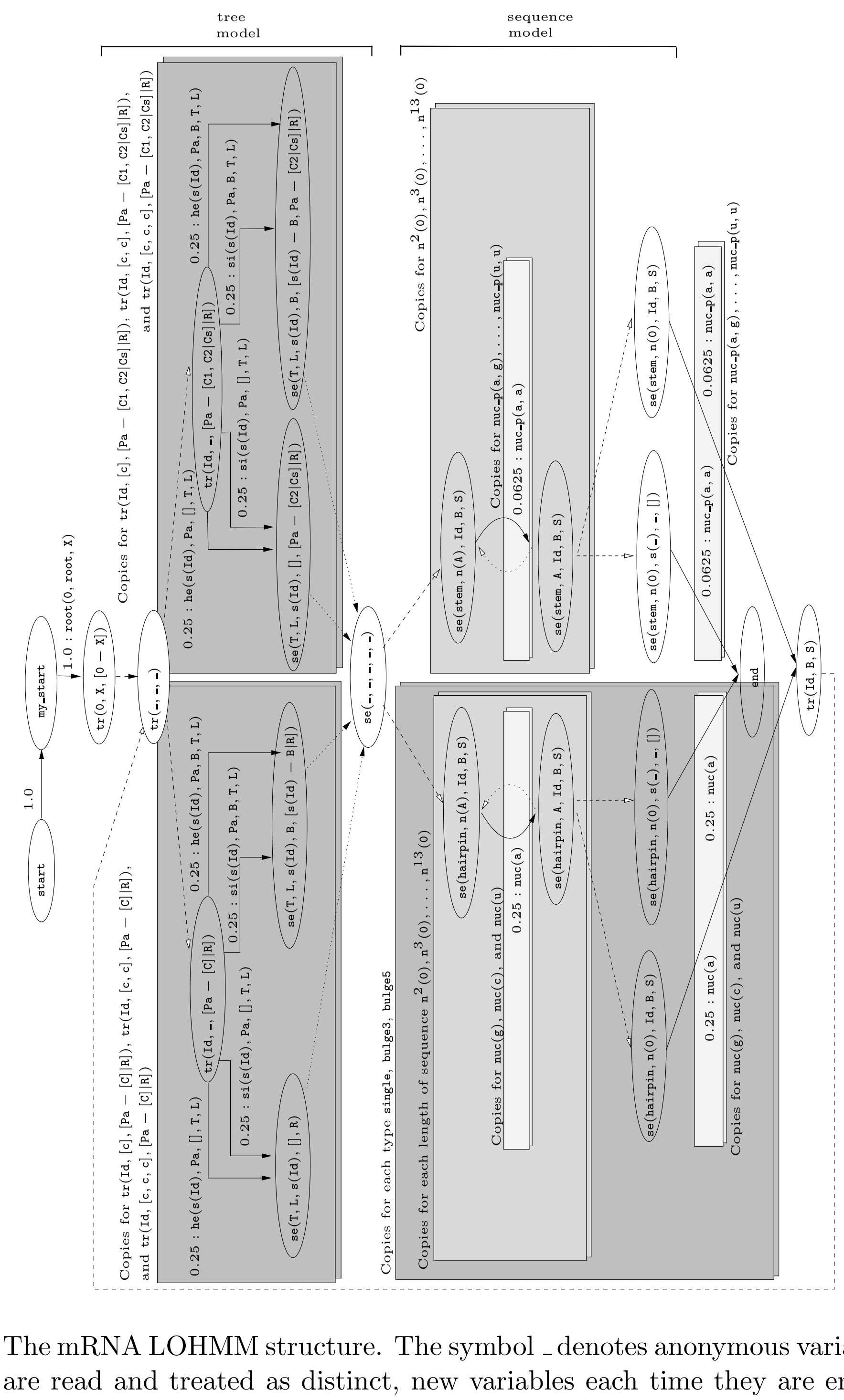

tree model
sequence model
start
1.0
my_start
1.0 : root(0, root, X)
tr(0, X, [0 − X])
tr(_, _, _)
Copies for tr(Id, [c], [Pa − [C]|R]), tr(Id, [c, c], [Pa − [C]|R]), and tr(Id, [c, c, c], [Pa − [C]|R])
Copies for tr(Id, [c], [Pa − [C1, C2|Cs]|R]), tr(Id, [c, c], [Pa − [C1, C2|Cs]|R]), and tr(Id, [c, c, c], [Pa − [C1, C2|Cs]|R])
0.25 : he(s(Id), Pa, [], T, L)
tr(Id, _, [Pa − [C]|R])
0.25 : he(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, [], T, L)
0.25 : si(s(Id), Pa, B, T, L)
se(T, L, s(Id), [], R)
se(T, L, s(Id), B, [s(Id) − B|R])
0.25 : he(s(Id), Pa, [], T, L)
tr(Id, _, [Pa − [C1, C2|Cs]|R])
0.25 : he(s(Id), Pa, B, T, L)
0.25 : si(s(Id), Pa, [], T, L)
0.25 : si(s(Id), Pa, B, T, L)
se(T, L, s(Id), [], [Pa − [C2|Cs]|R])
se(T, L, s(Id), B, [s(Id) − B, Pa − [C2|Cs]|R])
se(_, _, _, _, _)
Copies for each type single, bulge3, bulge5
Copies for each length of sequence n$^2$(0), n$^3$(0), . . . , n$^{13}$(0)
se(hairpin, n(A), Id, B, S)
Copies for nuc(g), nuc(c), and nuc(u)
0.25 : nuc(a)
se(hairpin, A, Id, B, S)
se(hairpin, n(0), Id, B, S)
se(hairpin, n(0), s(_), _, [])
0.25 : nuc(a)
0.25 : nuc(a)
Copies for nuc(g), nuc(c), and nuc(u)
Copies for n$^2$(0), n$^3$(0), . . . , n$^{13}$(0)
se(stem, n(A), Id, B, S)
Copies for nuc_p(a, g), . . . , nuc_p(u, u)
0.0625 : nuc_p(a, a)
se(stem, A, Id, B, S)
se(stem, n(0), s(_), _, [])
se(stem, n(0), Id, B, S)
0.0625 : nuc_p(a, a)
0.0625 : nuc_p(a, a)
Copies for nuc_p(a, g), . . . , nuc_p(u, u)
end
tr(Id, B, S)

Figure 9: The mRNA LOHMM structure. The symbol _ denotes anonymous variables which are read and treated as distinct, new variables each time they are encountered. There are copies of the shaded part. Terms are abbreviated using their starting alphanumerical; `tr` stands for `tree`, `se` for `structure_element`, `he` for `helical`, `si` for `single`, `nuc` for `nucleotide`, and `nuc_p` for `nucleotide_pair`.