A Characterization of the Dynamic Markov Compression FSM with Finite Conditioning Contexts

Suzanne Bunton[1]

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195

1 Introduction

The popular Dynamic Markov Compression Algorithm (DMC) [8] is a member of the family of data compression algorithms that combine an on-line stochastic data model with an arithmetic coder. DMC's distinguishing feature is an elegant but *ad hoc* modeling technique that provides state-of-the-art compression performance and matchless conceptual simplicity. In practice, however, the cost of DMC's simplicity and performance is often outrageous memory consumption. Several known attempts at reducing DMC's unwieldy model growth (e.g., [15, 19]) have rendered DMC's compression performance uncompetitive.

One reason why DMC's model growth problem has resisted solution is that the algorithm is poorly understood. DMC is the only published stochastic data model for which a characterization of its states, in terms of conditioning contexts, is unknown. Up until now, all that was certain about DMC was that a finite-context characterization exists, which was proved in [3] using a finiteness argument.

Here, we present and prove the first finite-context characterization of the states of DMC's data model. The impact of our characterization is threefold.

1. It proves that although DMC constructs a unifilar finite-order Markov FSM, DMC states cannot be uniquely characterized by single conditioning contexts, and therefore DMC models do not belong to the class of FSMX automata [13], which purportedly contain *all* Markovian FSMs.

2. It illuminates principled solutions for curbing counterproductive model growth.

3. It provides a sufficiently general on-line Markov model that can be parameterized to exactly emulate many other influential algorithms from the literature, including many popular FSMX models [7, 9, 11, 12, 13, 17, 18]. This allows

   (a) precise identification and comparison of the features that distinguish the structure and *a priori* statistical assumptions of different algorithms (such as "state selection," "blending," "update exclusions," etc.), and

[1]This work was supported in part by the National Science Foundation, Grant MIP-8858782, and by the Northwest Lab for Integrated Systems in the Department of Computer Science and Engineering at the University of Washington.

(b) experiments that evaluate the general effectiveness of specific model features by controlling the confounding factors induced by the myriad implementation decisions underlying any empirical evaluation.

Indeed, this work is preliminary to such a taxonomy and controlled empirical study [4].

Our analysis reveals that the DMC model, with or without its counterproductive portions, offers abstract structural features not found in other models. Ironically, the space-hungry DMC algorithm actually has a greater capacity for economical model representation than its counterparts have. Once identified, DMC's distinguishing features combine easily with the best features from other techniques. These combinations have the potential for achieving very competitive compression/memory tradeoffs.

2 The DMC Automaton

DMC constructs a series of finite-state machines (FSMs) $M = \{M_0, M_1, \ldots, M_m\}$, where for $0 \leq k \leq m$, $M_k = (S_k, A, \mu_k, s_0)$ such that

s_0 is the initial state;

S_k is the finite set of states, defined recursively as

$$\begin{aligned} S_k &= \{s_0\} \text{ if } k = 0, \\ &= S_{k-1} \cup \{s_k\}, \text{ otherwise;} \end{aligned}$$

A is the finite (input) alphabet; and

$\mu_k : S_k \times A \rightarrow S_k$ is the transition function, which extends to $\mu_k : S_k \times A^* \rightarrow S_k$ in the traditional fashion.[2]

As each symbol in a given input sequence is scanned, DMC applies a refinement eligibility criterion, $\mathcal{E} : M \times A^* \rightarrow \{T, F\}$, to the current FSM, M_k, to decide if it should be extended into M_{k+1}.

The FSMs $M_k, \forall k \geq 0$ are defined recursively. At the start, $\mu_0(s_0, a) = s_0, \forall a \in A$. That is, the initial model consists of the single state, s_0 and a reflexive transition for each symbol in the input alphabet.[3] During processing of an input sequence, whenever $\mathcal{E}(M_k, wa) = T$, a new machine M_{k+1} is constructed from M_k as follows. Let $s_p = \mu_k(s_0, w)$ and $s_t = \mu_k(s_p, a)$. Then, we define

$$\mu_{k+1}(s_p, a) = s_{k+1} \quad (1)$$

$$\mu_{k+1}(s_p, b) = \mu_k(s_p, b), \forall b \in A - \{a\} \quad (2)$$

$$\mu_{k+1}(s_i, b) = \mu_k(s_i, b), \forall b \in A, \forall i \leq k, i \neq p \quad (3)$$

$$\mu_{k+1}(s_{k+1}, b) = \mu_{k+1}(s_t, b), \forall b \in A. \quad (4)$$

The process of extending the current FSM M_k to M_{k+1}, called "cloning," is depicted in Figure 1. Cloning [8] simply redirects the current transition, $\mu_k(s_p, a)$, to a newly added state s_{k+1}. Cloning is performed whenever $\mathcal{E}(M_k, wa) = T$, where w is the already scanned portion of the input sequence. The newly added state is a copy of the transition's destination such that the transitions leaving the new state are copied

[2] Throughout, we shall denote the concatenation of strings w and y as wy or as $w \cdot y$.

[3] Other possible initial models are given in [8].

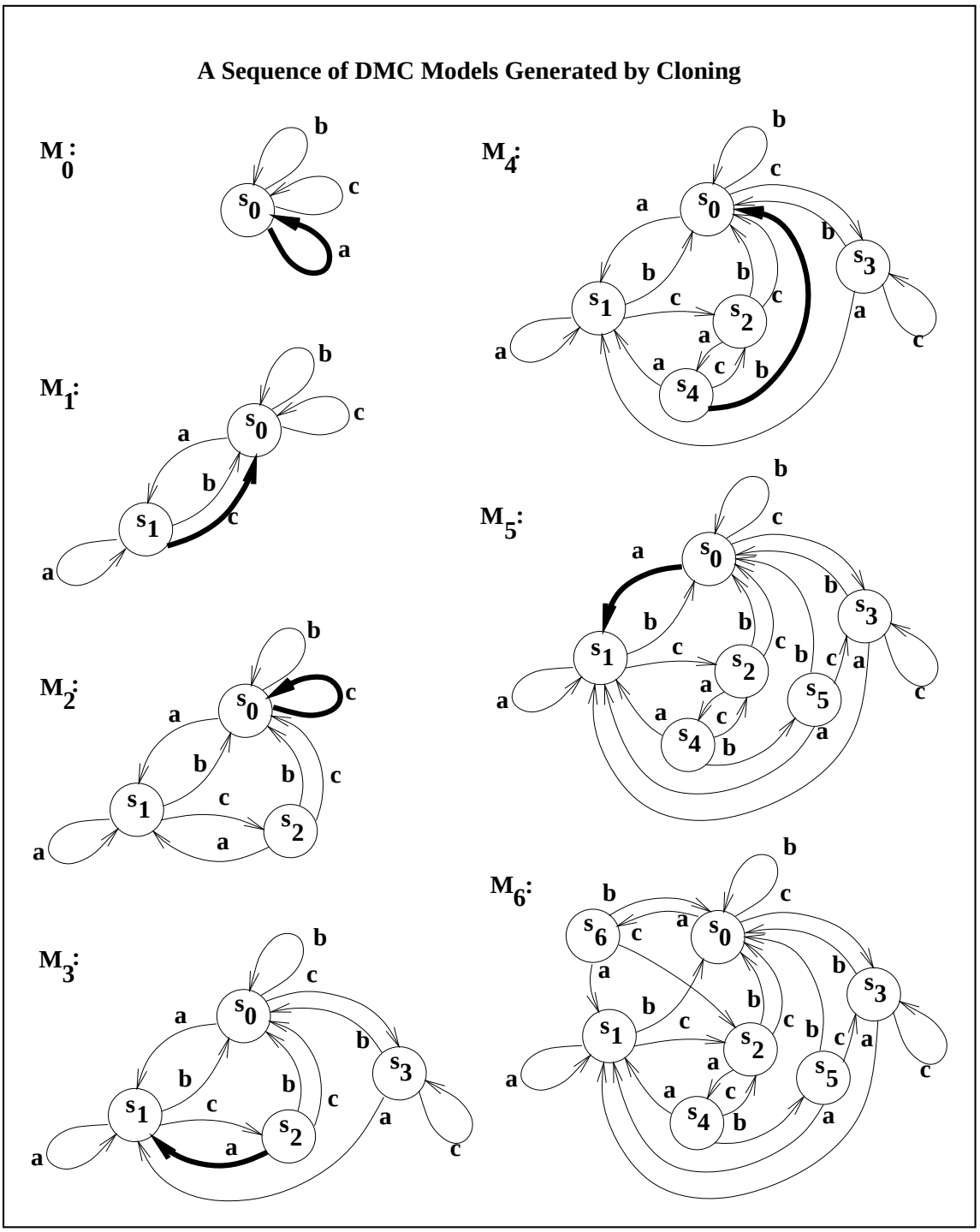

Figure 1: **DMC's Finite-State Data Model.** DMC's finite-state data model is created incrementally by cloning the destination of the current transition, if it is determined to be eligible. The **bold** (presumably eligible) transition in each model M_k is redirected to a newly added state s_{k+1} to form model M_{k+1}. The transitions leaving s_{k+1} are copied from the **bold** transition's former destination, after the **bold** transition is redirected.

from the transitions leaving the original state. The number of models in M, and the number of states in the final FSM, M_m, equal $m+1$. The number of proper prefixes wa of the given input sequence for which the criterion $\mathcal{E}(M_k, wa)$ holds determines the value of m.

The eligibility criterion given in [8] is based upon the popularity of the current transition relative to the popularity of its destination. That is, $\mathcal{E}()$ is equivalent to the well-formed formula

$$t_1 \leq |\{y : y \in \text{ prefixes}(w), \mu_k(s_0, y) = \mu_k(s_0, w), \mu_k(s_0, ya) = \mu_k(s_0, wa)\}| \text{ and}$$
$$t_2 \leq |\{y : y \in \text{ prefixes}(w), \mu_k(s_0, y) \neq \mu_k(s_0, w), \mu_k(s_0, ya) = \mu_k(s_0, wa)\}|,$$

where thresholds t_1 and t_2 are algorithm input parameters. This particular definition of $\mathcal{E}()$ causes DMC to construct a stochastic model that cannot converge to any finite stochastic source that could be assumed to have emitted the sequence. The following analysis of DMC's model structure does not depend upon any particular definition of $\mathcal{E}()$.

3 Observable Structure in DMC

3.1 Definitions

The following definitions formulate the intuition and axioms of our analysis. They are illustrated in Figure 2.

$$\begin{aligned}
\mathbf{prefix}(s_i) &= s_0,\ i = 0 \\
&= s_p : \mu_{i-1}(s_p, a) \neq \mu_i(s_p, a) = s_i, p < i, a \in A,\ i > 0 \\
\mathbf{symbol}(s_i) &= \lambda,\ i = 0,\ \text{where } \lambda \text{ denotes the empty string} \\
&= a : \mu_i(\mathbf{prefix}(s_i), a) = s_i,\ a \in A,\ i > 0 \\
\mathbf{context}(s_i) &= \lambda,\ i = 0 \\
&= \mathbf{context}(\mathbf{prefix}(s_i))\mathbf{symbol}(s_i),\ i > 0 \\
\mathbf{suffix}(s_i) &= s_0,\ i = 0 \\
&= \mu_{i-1}(\mathbf{prefix}(s_i), \mathbf{symbol}(s_i)),\ i > 0 \\
\mathbf{extns}_k(s_i) &= \{s_d : \mathbf{suffix}(s_d) = s_i,\ d \leq k\} \\
\mathbf{extns}^*_k(s_i) &= \mathbf{extns}_k(s_i) \cup \textstyle\bigcup_{s_d \in \mathbf{extns}_k(s_i)} \mathbf{extns}^*_k(s_d)
\end{aligned}$$

The function $\mathbf{prefix} : S \rightarrow S$ is used with the function $\mathbf{symbol} : S \rightarrow A$ to recursively map states to finite strings, or contexts. The state $\mathbf{prefix}(s_i)$ is the source of the transition that was redirected to s_i when s_i was added to the model. The character $\mathbf{symbol}(s_i)$ labels the transition that was originally redirected to state s_i, and any subsequently added transitions into s_i. Lemma 3.1 proves that any string that brings M_k into state s_i ends in the finite string $\mathbf{context}(s_i)$, for $k \geq i$.

The function $\mathbf{suffix} : S \rightarrow S$ organizes the states of M_k into a tree, acting as a parent pointer. The state $\mathbf{suffix}(s_i)$ is the former destination of the transition that was redirected to state s_i when s_i was created. In the tree induced by M_k, the state s_i is the parent of the states in $\mathbf{extns}_k(s_i)$. Equivalently, the set of states $\mathbf{extns}_k(s_i)$ equals the children of state s_i. The name "**extns**" is used instead of "children" because the relationships among the conditioning contexts associated with each state, rather than their positions in the tree, are the primary points of interest.

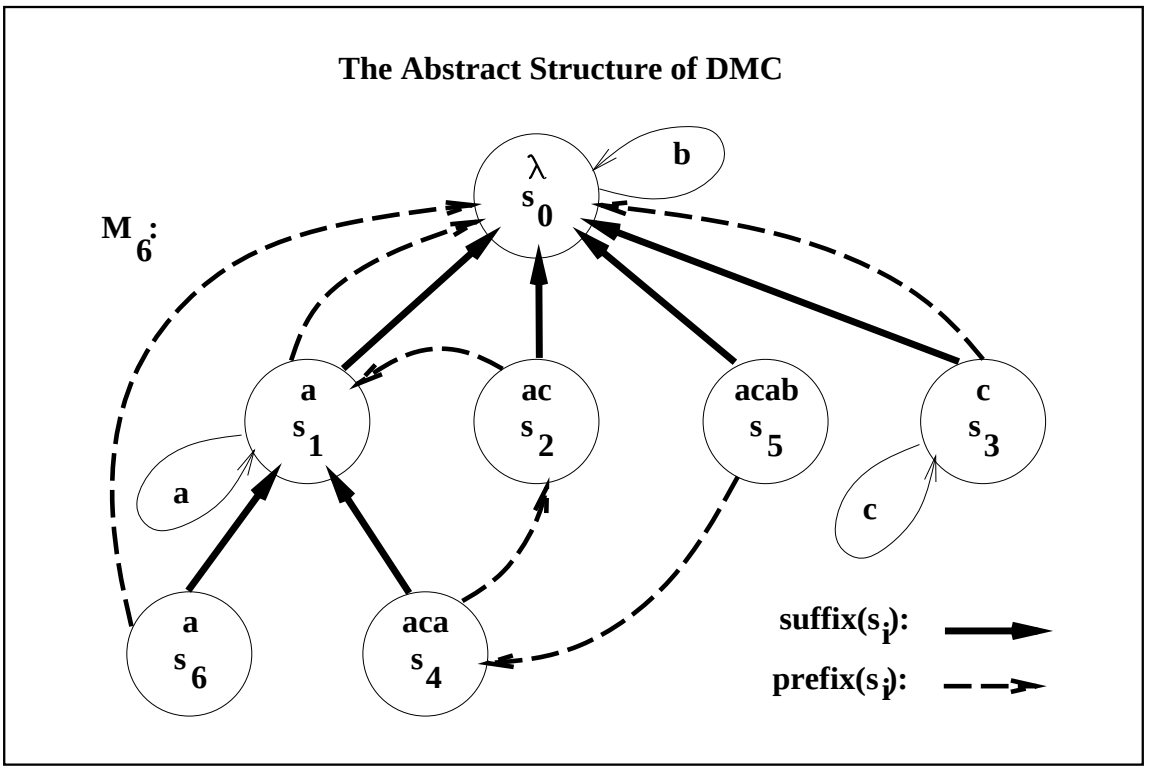

Figure 2: **Observable Structure in DMC Models.** For any state s_i, **suffix**(s_i) is the original destination of the transition that was redirected to s_i when s_i was created; **prefix**(s_i) is the source of the transition which was redirected to s_i, when s_i was added to the model; and **symbol**(s_i) labels the transition that was originally redirected to s_i, and any subsequently added transitions into s_i. The context of s_i, **context**(s_i), labels each state.

The non-reflexive transitions of model M_6, pictured in Figure 1, are omitted. However, the reflexive transitions of M_6 are included here to illustrate the consistent substructures they define in the DMC model. There are always $|A|$ reflexive transitions in the model. (Here $A = \{a, b, c\}$). When a reflexive transition is redirected by cloning, the newly added state will have a reflexive transition with the same symbol. For any state s_i with a reflexive transition (it can only have one, if $i \neq 0$), **context**(s_i) will be a sequence consisting of a finite number of occurrences of the symbol on the reflexive edge, which equals the state's **symbol**. For example, if the reflexive edge labeled c is redirected to a new state s_7, **prefix**(s_7) will equal s_3 and **context**(s_7) will equal cc.

The state s_6 is an example of a state that has been created by redirecting a *prefix transition*. That is, the original edge which was redirected to point to s_1 was redirected again when s_6 was added to the model. Both s_6 and s_1 therefore have identical **context**s.

The closure of the set of children of state s_i equals the set of descendants of s_i, and is denoted $\mathbf{extns}^*_k(s_i)$.

Additionally, two observations from [3] are used repeatedly in the proofs of the lemmas and theorems to follow, and they can be restated using the **suffix**function above. The observations basically point out that the only states whose input transitions are affected when s_{k+1} is added are $\mathbf{suffix}(s_{k+1})$ and s_{k+1}.

Observation 3.1

$$\begin{aligned} &\forall i \leq k, w \in A^*, \\ &\mu_{k+1}(s_i, w) \neq s_{k+1} \quad \Rightarrow \quad \mu_{k+1}(s_i, w) = \mu_k(s_i, w). \end{aligned}$$

Observation 3.2

$$\begin{aligned} &\forall i \leq k+1, w \in A^*, \\ &\mu_k(s_i, w) \neq \mu_{k+1}(s_i, w) \quad \Rightarrow \quad \mu_k(s_i, w) = \mathbf{suffix}(s_{k+1}) \textit{ and} \\ &\qquad\qquad\qquad\qquad\qquad\quad \mu_{k+1}(s_i, w) = s_{k+1}. \end{aligned}$$

3.2 Contexts of DMC States

Although no semantics have been assigned to DMC states in the related literature, the evocative names of the above functions do imply accurate semantics. That is, the following properties hold for all s_i:

- [Prefixes] $\mathbf{context}(\mathbf{prefix}(s_i))$ is a prefix of $\mathbf{context}(s_i)$,
- [Suffixes] $\mathbf{context}(\mathbf{suffix}(s_i))$ is a suffix of $\mathbf{context}(s_i)$,
- [Extensions] $s_d \in \mathbf{extns}^*_k(s_i)$ implies that $\mathbf{context}(s_d)$ extends $\mathbf{context}(s_i)$ on the left by zero or more symbols.

Note that $\mathbf{context}(\mathbf{prefix}(s_i))$ is exactly one symbol shorter than $\mathbf{context}(s_i)$, while $\mathbf{context}(s_i)$ may be arbitrarily longer than $\mathbf{context}(\mathbf{suffix}(s_i))$. Thus DMC builds a context model with variable-length *minimal extensions*[4] of a context. Furthermore, if the eligibility criterion $\mathcal{E}()$ allows transitions to be redirected more than once, as DMC's does, it is possible that $\mathbf{context}(\mathbf{suffix}(s_i)) = \mathbf{context}(s_i)$.

The **context**() of a state in a DMC model is provably analogous the context of a state in an FSMX model [13]. The next lemma shows that every string entering a given state s_i in model M_k ends in the substring $\mathbf{context}(s_i)$.

Lemma 3.1 (Context Lemma)

$$\begin{aligned} &\forall k, \ \forall i \textit{ and } j \leq k, \ \forall wa \in A^+, \\ &\qquad\qquad \mu_k(s_j, wa) = s_i \quad \Rightarrow \quad wa \in A^*\mathbf{context}(s_i). \end{aligned}$$

The proof is by induction on k. The base case, $k = 0$, is trivial because $\mathbf{context}(s_0) = \lambda$. To prove the induction step, assume the statement holds for $\mu_k()$ and assume that $\mu_{k+1}(s_j, wa) = s_i$. We can further assume that $j \leq k$ because $\mu_{k+1}(s_{k+1}, wa) = \mu_{k+1}(\mathbf{suffix}(s_{k+1}), wa)$ and $\mathbf{suffix}(s_{k+1}) \neq s_{k+1}$.

[4] A *minimal extension* is the $y \in A^*$ *s.t.* $\mathbf{context}(s_i) = y \cdot \mathbf{context}(\mathbf{suffix}(s_i))$.

There are two cases to consider, $i \neq k+1$, and $i = k+1$. If $s_i \neq s_{k+1}$, then $\mu_{k+1}(s_{k+1}, wa) = \mu_k(s_j, wa)$. So, by the induction hypothesis, $wa \in A^*\mathbf{context}(s_{k+1})$. For the case that $i = k+1$, let $a = \mathbf{symbol}(s_{k+1})$ and suppose that

$$\mu_{k+1}(s_j, wa) = \mu_{k+1}(\mu_{k+1}(s_j, w), a) = s_{k+1}.$$

Then, by definition of M_{k+1},

$$\mu_k(s_j, wa) = \mu_k(\mu_k(s_j, w), a) = \mathbf{suffix}(s_{k+1}).$$

Also, by inspection of the definition of M_{k+1}, we know that

$$\mu_{k+1}(s_j, w) \in \{s_{k+1},\ \mathbf{prefix}(s_{k+1})\}.$$

$\mu_{k+1}(s_j, w) = \mathbf{prefix}(s_{k+1})$

$\Rightarrow$	$\mu_{k+1}(s_j, w) = \mu_k(s_j, w) = \mathbf{prefix}(s_{k+1})$	by Observation 3.1
$\Rightarrow$	$w \in A^*\mathbf{context}(\mathbf{prefix}(s_{k+1}))$	by ind. hyp.
$\Rightarrow$	$wa \in A^*\mathbf{context}(\mathbf{prefix}(s_{k+1}) \cdot a$	by regular set concatenation
$\Rightarrow$	$wa \in A^*\mathbf{context}(s_{k+1})$	by def **context**().

$\mu_{k+1}(s_j, w) = s_{k+1}$

$\Rightarrow$	$\mu_{k+1}(s_{k+1}, a) = s_{k+1}$	since $\mu_{k+1}(s_j, wa) = s_{k+1}$
$\Rightarrow$	$\mathbf{suffix}(s_{k+1}) = \mathbf{prefix}(s_{k+1})$ and $\mu_k(\mathbf{suffix}(s_{k+1}), a) = \mathbf{suffix}(s_{k+1})$	by Lemma 3.2
$\Rightarrow$	$\mu_k(s_j, w) = \mathbf{suffix}(s_{k+1})$	by Observation 3.2
$\Rightarrow$	$\mu_k(s_j, w) = \mathbf{prefix}(s_{k+1})$	by subst.
$\Rightarrow$	$w \in A^*\mathbf{context}(\mathbf{prefix}(s_{k+1}))$	by ind. hyp.
$\Rightarrow$	$wa \in A^*\mathbf{context}(\mathbf{prefix}(s_{k+1}) \cdot a$	by regular set concatenation
$\Rightarrow$	$wa \in A^*\mathbf{context}(s_{k+1})\square$	

3.3 Reflexive Edges in DMC

Reflexive edges appear only under certain circumstances in DMC models, and any M_k will have reflexive edges if and only if the initial state s_0 does. Intuitively, Lemma 3.2 states that any other reflexive edges can only be created by redirecting reflexive edges. When a reflexive edge is redirected, the **prefix** of the new destination state equals that state's **suffix**. Over time, a reflexive edge will either stay the same or will point down into its source state's extensions (subtree). Conversely, any state with a self-loop, or an edge that points down into its subtree, was initially added to the model as as the new destination of a redirected reflexive edge.

Only the base case of the following lemma, that is, a description of reflexive edges leaving any novel state s_k in model M_k, is required for the proof of the main result, the complete characterization of DMC's structure. However, by proving the lemma for all states i in any model M_k, we can describe the behavior of all reflexive edges in DMC models over time. Incidentally, when $i = 0$, the left hand side of the equivalence holds trivially. That is, all edges leaving the root state s_0 enter either s_0 or a node in its subtree.

Lemma 3.2 (Reflexive Edge Characterization)

$$\begin{array}{l} \forall k \geq 1,\ \forall i, 1 \leq i \leq k,\ \forall a \in A, \\ \quad \mu_k(s_i, a) \in \{s_i\} \cup \mathbf{extns}_k^*(s_i) \quad \Leftrightarrow \quad \mathbf{suffix}(s_i) = \mathbf{prefix}(s_i) \textit{ and} \\ \qquad\qquad\qquad\qquad\qquad\quad \mu_{i-1}(\mathbf{suffix}(s_i), a) = \mathbf{suffix}(s_i) \end{array}$$

The proof follows below by induction on $k - i$.

To prove the induction base, where $k = i$ and

$$\begin{array}{rcl} \mu_k(s_k, a) \in \{s_k\} \cup \mathbf{extns}_k^*(s_k) & \Leftrightarrow & \mathbf{suffix}(s_k) = \mathbf{prefix}(s_k) \text{ and} \\ & & \mu_{k-1}(\mathbf{suffix}(s_k), a) = \mathbf{suffix}(s_k), \end{array}$$

we first recall from the definition of M_k that

$$\begin{array}{rcll} \mu_{k-1}(\mathbf{prefix}(s_k), a) & = & \mathbf{suffix}(s_k), & (5) \\ \mu_k(\mathbf{prefix}(s_k), a) & = & s_k, & (6) \\ & \text{and} & & \\ \forall b \in A,\ \mu_k(s_k, b) & = & \mu_k(\mathbf{suffix}(s_k), b). & (7) \end{array}$$

The proof is straightforward:

$\mu_{k-1}(\mathbf{suffix}(s_k), a) = \mathbf{suffix}(s_k)$ and $\mathbf{prefix}(s_k) = \mathbf{suffix}(s_k)$

$\Rightarrow$ $\mu_{k-1}(\mathbf{suffix}(s_k), a) \neq \mu_k(\mathbf{suffix}(s_k), a)$ and
$\mu_{k-1}(\mathbf{prefix}(s_k), a) \neq \mu_k(\mathbf{prefix}(s_k), a)$ by (5), (6), subst.

$\Rightarrow$ $\mu_k(\mathbf{suffix}(s_k), a) = s_k$ by Observation 3.2

$\Rightarrow$ $\mu_k(s_k, a) = s_k$ by (7), subst.

$\Rightarrow$ $\mu_k(s_k, a) \in \{s_k\} \cup \mathbf{extns}_k^*(s_k)$ $\mathbf{extns}_k^*(s_k) = \{\}$.

$\mu_k(s_k, a) \in \{s_k\} \cup \mathbf{extns}_k^*(s_k)$

$\Rightarrow$ $\mu_k(s_k, a) = s_k$ $\mathbf{extns}_k^*(s_k) = \{\}$.

$\Rightarrow$ $\mu_k(\mathbf{suffix}(s_k), a) = s_k$ by (7), subst.

$\Rightarrow$ $\mu_{k-1}(\mathbf{suffix}(s_k, a) \neq \mu_k(\mathbf{suffix}(s_k, a)$ since $s_k \notin S_{k-1}$

$\Rightarrow$ $\mu_{k-1}(\mathbf{suffix}(s_k, a) = \mathbf{suffix}(s_k)$ by Observation 3.2

$\Rightarrow$ $\mu_{k-1}(\mathbf{suffix}(s_k), a) = \mu_{k-1}(\mathbf{prefix}(s_k), a)$, and
$\mu_k(\mathbf{suffix}(s_k), a) = \mu_k(\mathbf{prefix}(s_k), a)$ by (5) and (6), subst.

$\Rightarrow$ $\mu_{k-1}(\mathbf{prefix}(s_k, a) \neq \mu_k(\mathbf{prefix}(s_k, a)$ subst

$\Rightarrow$ $\mathbf{prefix}(s_k) = \mathbf{suffix}(s_k)$ since only one transition in M_{k-1} changed to form M_k.

$\Rightarrow$ $\mu_{k-1}(\mathbf{suffix}(s_k), a) = \mathbf{suffix}(s_k)$ and
$\mathbf{prefix}(s_k) = \mathbf{suffix}(s_k)$.

For the induction step, assume the statement holds for a given state $s_i, i > 0$, in model M_k and consider the same state s_i in model M_{k+1}. Note that $s_i \neq s_{k+1}$ because $s_{k+1} \notin S_k$.

When $\mu_{k+1}(s_i, a) \in \mathbf{extns}_{k+1}^*(s_i) \cup \{s_i\}$, there are three mutually exclusive cases to consider: $\mu_{k+1}(s_i, a) = s_i$, $\mu_{k+1}(s_i, a) \in \mathbf{extns}_k^*(s_i)$, and $\mu_{k+1}(s_i, a) = s_{k+1}$.

$\mu_{k+1}(s_i, a) = s_i$
$\Rightarrow$ $\mu_k(s_i, a) = s_i$ by Observation 3.1
$\Rightarrow$ $\mathbf{suffix}(s_i) = \mathbf{prefix}(s_i)$ and
$\mu_{i-1}(\mathbf{suffix}(s_i), a) = \mathbf{suffix}(s_i)$ by ind. hyp.

$\mu_{k+1}(s_i, a) \in \mathbf{extns}^*_k(s_i)$
$\Rightarrow$ $\mu_{k+1}(s_i, a) \neq s_{k+1}$ $s_{k+1} \notin S_k$
$\Rightarrow$ $\mu_{k+1}(s_i, a) = \mu_k(s_i, a)$ by Observation 3.1
$\Rightarrow$ $\mathbf{suffix}(s_i) = \mathbf{prefix}(s_i)$ and
$\mu_{i-1}(\mathbf{suffix}(s_i), a) = \mathbf{suffix}(s_i)$ by ind. hyp., subst.

$\mu_{k+1}(s_i, a) = s_{k+1}$
$\Rightarrow$ $\mu_{k+1}(s_i, a) \neq \mu_k(s_i, a)$ $s_{k+1} \notin S_k$
$\Rightarrow$ $\mu_k(s_i, a) = \mathbf{suffix}(s_{k+1})$ by Obs. 3.2
$\Rightarrow$ $\mu_k(s_i, a) \in \mathbf{extns}^*_{k+1}(s_i) - \{s_{k+1}\}$ $\mu_k(s_i, a) \in \mathbf{extns}^*_{k+1}(s_i)$ and $\mu_k(s_i, a) \neq s_{k+1}$
$\Rightarrow$ $\mu_k(s_i, a) \in \mathbf{extns}^*_k(s_i)$ $\mathbf{extns}^*_k(s_i) = \mathbf{extns}^*_{k+1}(s_i) - \{s_{k+1}\}$
$\Rightarrow$ $\mathbf{suffix}(s_i) = \mathbf{prefix}(s_i)$ and
$\mu_{i-1}(\mathbf{suffix}(s_i), a) = \mathbf{suffix}(s_i)$ by ind. hyp.

When $\mathbf{suffix}(s_i) = \mathbf{prefix}(s_i)$ and $\mu_{i-1}(\mathbf{suffix}(s_i), a) = \mathbf{suffix}(s_i)$, we know by the induction hypothesis that $\mu_k(s_i, a) \in \mathbf{extns}^*_k(s_i) \cup \{s_i\}$. Either $\mu_{k+1}(s_i, a) = \mu_k(s_i, a)$, and so $\mu_{k+1}(s_i, a) \in \mathbf{extns}^*_{k+1}(s_i) \cup \{s_i\}$ because $\mathbf{extns}^*_k(s_i) \subseteq \mathbf{extns}^*_{k+1}(s_i)$; or $\mu_{k+1}(s_i, a) \neq \mu_k(s_i, a)$. In the latter case,

$\mu_{k+1}(s_i, a) \neq \mu_k(s_i, a)$
$\Rightarrow$ $\mu_k(s_i, a) = \mathbf{suffix}(s_{k+1})$ by Observation 3.2
$\Rightarrow$ $\mathbf{suffix}(s_{k+1}) \in \mathbf{extns}^*_k(s_i) \cup \{s_i\}$ by ind. hyp., subst.
$\Rightarrow$ $s_{k+1} \in \mathbf{extns}^*_{k+1}(s_i)$ by def. $\mathbf{extns}^*()\Box$

Lemma 3.2 implies the following regularities for DMC models, assuming the initial model given earlier:

- For each $b \in A$ there exists exactly one reflexive edge labeled b.
- No state other than s_0 may have more than one reflexive edge.
- And, if a state s_i has a reflexive edge labeled a, then $\mathbf{context}(s_i) = a^h$, where h is the length of the path of $\mathbf{prefix}()$ pointers from s_i to s_0.

Lastly, the following technical corollary to Lemma 3.2 will be required to prove the upcoming DMC characterization. If a string brings a model M_{k+1} into the new state s_{k+1} and it also brought the preceding model M_k to the **prefix**state of the new state, then the new state must have a reflexive out-edge, labeled with the state's **symbol**.

Corollary 3.1 (to Reflexive Edge Characterization)

$$\forall k, \forall j \leq k, \forall w \in A^*,$$
$$\mu_k(s_j, w) = \mathbf{prefix}(s_{k+1}) \textit{ and } \mu_{k+1}(s_j, w) = s_{k+1} \;\Rightarrow\; \mu_{k+1}(s_{k+1}, \mathbf{symbol}(s_{k+1})) = s_{k+1}.$$

Proof:

$\mu_k(s_j, w) = \mathbf{prefix}(s_{k+1})$ *and* $\mu_{k+1}(s_j, w) = s_{k+1}$

$\Rightarrow$ $\mu_k(s_j, w) \neq \mu_{k+1}(s_j, w)$ $\quad \mathbf{prefix}(s_{k+1}) \neq s_{k+1}$

$\Rightarrow$ $\mu_k(s_j, w) = \mathbf{suffix}(s_{k+1})$ $\quad$ *by Observation 3.2*

$\Rightarrow$ $\mathbf{prefix}(s_{k+1}) = \mathbf{suffix}(s_{k+1})$ $\quad$ *subst.*

$\Rightarrow$ $\mu_k(\mathbf{prefix}(s_{k+1}), \mathbf{symbol}(s_{k+1})) = \mathbf{suffix}(s_{k+1})$ $\quad$ *by def.* $\mathbf{suffix}(s_{k+1})$

$\Rightarrow$ $\mu_{k+1}(\mathbf{prefix}(s_{k+1}), \mathbf{symbol}(s_{k+1})) = s_{k+1}$ $\quad$ *by def.* $\mu_{k+1}()$

$\Rightarrow$ $\mu_k(\mathbf{suffix}(s_{k+1}), \mathbf{symbol}(s_{k+1})) = \mathbf{suffix}(s_{k+1})$ $\quad$ *subst.*

$\Rightarrow$ $\mu_{k+1}(s_{k+1}, \mathbf{symbol}(s_{k+1})) = \{s_{k+1}\} \cup \mathbf{extns}^*_{k+1}(s_{k+1})$ $\quad$ *by Lemma 3.2, contrap.*

$\Rightarrow$ $\mu_{k+1}(s_{k+1}, \mathbf{symbol}(s_{k+1})) = s_{k+1}$ $\quad \mathbf{extns}^*_{k+1}(s_{k+1}) = \{\}$ □

4 A Finite-Order Characterization of DMC States

A finite-state stochastic model, such as DMC's, is traditionally defined in terms of its *structure* and *parameters*. The language of an individual FSM state is the set of strings which put the FSM into the given state, by following the successive transitions labelled with each symbol of the string left-to-right, starting at the FSM's initial state. The set consisting of each language of each state in a stochastic model forms a partition on the set of possible input sequences to the model, that is, a *context partition*. The *structure* of a stochastic model is defined by its context partition.

Thus each state in the model is associated with a single class of strings, and the model is used to successively classify each progressively longer prefix of an entire input sequence. To use such a model to estimate probabilities of a sequence, or to predict or generate such a sequence, each state in the model must also be associated with a (usually empirical) probability measure on the symbols that may immediately follow any string belonging to the state's class. DMC's solution to this *parameterization* problem is described in Section 5.3.

DMC is unique in the data compression literature, in that its model was not originally defined to represent a given context partition. The goal of this analysis is to identify the (hitherto unknown) context partition of any DMC model, so that aspects of the DMC technique may be meaningfully compared with features of other techniques in the literature. DMC's context partition is given below by the function $C_k : S \rightarrow 2^{A^*}$, which, as we prove, describes the language of any state in a DMC model M_k.

Theorem 4.1 (Characterization of DMC Structure)

$$\begin{aligned} \textit{Let} \quad C_k(s_i) &= \mathbf{L}(s_i) - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} \mathbf{L}(s_d), \qquad (8) \\ \textit{where} \ \mathbf{L}(s_i) &= A^*, \ \textit{if} \quad i = 0 \\ &= C_{i-1}(\mathbf{prefix}(s_i)) \cdot \mathbf{symbol}(s_i), \quad \textit{otherwise.} \end{aligned}$$

Then, $\forall k, \ \forall i \leq k, \quad$ *and* $\forall w \in A^*$,

$$w \in C_k(s_i) \quad \Leftrightarrow \quad \mu_k(s_0, w) = s_i. \qquad (9)$$

Before we prove that the characterization $C_k()$ is correct, a discussion of some properties of the functions introduced above will provide some insight.

The function $C_k : S_k \rightarrow 2^{A^*}$ maps each state to a set of conditioning contexts. Equation (9) implies that the sets of distinct states are disjoint; that is, $C_k()$ relies on the function $\mathbf{L}()$, which is one-to-one regardless of the definition of $\mathcal{E}()$. The regular set $\mathbf{L}(s_i)$ precisely describes the set of strings that bring M_k to state s_i or to any descendant of s_i. Note that although the language $C_k(s_i)$ of a state s_i changes with k (which is monotone increasing), the language of the subtree rooted by a state s_i, that is, $\mathbf{L}(s_i)$, does not (even though the subtree itself may change in structure).

The set $\mathbf{L}(s_i) \subseteq A^*\mathbf{context}(s_i)$, and recursive expansions of the regular expression $\mathbf{L}(s_i)$ quit branching at all states s_p such that $\mathbf{L}(s_p) = A^*\mathbf{context}(s_p)$. Figure 3 shows the DMC model substructures relevant to any arbitrary state s_i, and illustrates the recursive expansion of expressions $C_k(s_i)$ and $\mathbf{L}(s_i)$.

The function $\mathbf{L}()$ is well-defined, and would still be well-defined even if we did not require the recursive expansion to continue down to the initial state s_0. This can be accomplished by optimizing the terminating expansion given above:

$$\begin{aligned}\mathbf{L}(s_i) &= A^* \cdot \mathbf{context}(s_i), \text{ if } i = 0 \text{ or } \forall j \leq i \\ &\quad \mathbf{prefix}(s_j) \neq \mathbf{prefix}(\mathbf{suffix}(s_j)) \text{ and } \mathbf{extns}_{j-1}(\mathbf{prefix}(s_j)) = \{\} \quad (10) \\ &= C_{i-1}(\mathbf{prefix}(s_i)) \cdot \mathbf{symbol}(s_i), \text{ otherwise.}\end{aligned}$$

The resulting regular expressions for the languages of each state are the same with either definition, but the terminating criterion used in Equation (10) above creates the shallowest recursive expansion. Furthermore, the model condition required by Equation (10), the base of the inductive definition, describes the exact requirements for the states of a DMC model to be characterizable by single finite strings.

4.1 Correctness Proof of the DMC Characterization

Here we prove Theorem 4.1, which states that Equation (8) characterizes the partition of conditioning contexts that is induced by the states of any model M_k, by proving that $C_k : S \rightarrow 2^{A^*}$ describes the language of each state in M_k, for all k. The proof proceeds by induction on k, the number of states that have been added to M_0 to create the k progressive refinements to M_0 which result in model M_k.

The induction base is trivial: $\mu_0(s_0, w) = s_0$ and $w \in C_0(s_0) = A^*\lambda$. $C_0(s_0) = A^*\lambda$ because $\mathbf{extns}_0(s_0) = \{\}$ and $\mathbf{L}(s_0) = A^*\lambda$.

For the induction step we assume that

$$\begin{aligned}&\forall h \leq k,\ \forall i \leq h, \text{ and } \forall w \in A^*, \\ &\qquad w \in C_h(s_i) \ \Leftrightarrow \ \mu_h(s_0, w) = s_i,\end{aligned}$$

and prove that

$$\begin{aligned}&\forall i \leq k+1, \text{ and } \forall w \in A^*, \\ &\qquad w \in C_{k+1}(s_i) \ \Leftrightarrow \ \mu_{k+1}(s_0, w) = s_i.\end{aligned}$$

When $w = \lambda$, we know that $w \in C_{k+1}(s_i) \Leftrightarrow s_i = s_0 \Leftrightarrow \mu_{k+1}(s_0, w) = s_i$. This equivalence follows from the fact that $s_d \in \mathbf{extns}(s_i) \Rightarrow \mathbf{symbol}(s_d) = \mathbf{symbol}(s_i)$ and the definition of $C_k()$, which expands to form

$$C_k(s_0) = A^* - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} C_{d-1}(\mathbf{prefix}(s_d)) \cdot \mathbf{symbol}(s_d), \text{and}$$

$$C_k(s_{i\neq 0}) = \left(C_{i-1}(\mathbf{prefix}(s_i)) - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} C_{d-1}(\mathbf{prefix}(s_d)) \right) \cdot \mathbf{symbol}(s_i).$$

Thus, $C_k(s_0)$ contains λ and $C_k(s_{i\neq 0})$ only contains strings ending in $\mathbf{symbol}(s_i)$.

When $w \neq \lambda$, we let $w = va$ and consider two cases: $s_i = s_{k+1}$ and $s_i \neq s_{k+1}$. The complete proofs for each case are given below.

4.1.1 Proof that $va \in C_{k+1}(s_{k+1}) \Leftrightarrow \mu_{k+1}(s_0, va) = s_{k+1}$

$va \in C_{k+1}(s_{k+1})$
- $\Leftrightarrow$ $va \in \mathbf{L}(s_{k+1}) - \bigcup_{s_d \in \mathbf{extns}_{k+1}(s_{k+1})} \mathbf{L}(s_d)$ — by def. $C_{k+1}()$
- $\Leftrightarrow$ $va \in \mathbf{L}(s_{k+1})$ — $\mathbf{extns}_{k+1}(s_{k+1}) = \{\}$
- $\Leftrightarrow$ $va \in C_k(\mathbf{prefix}(s_{k+1})) \cdot \mathbf{symbol}(s_{k+1})$ — by def. $\mathbf{L}()$
- $\Leftrightarrow$ $v \in C_k(\mathbf{prefix}(s_{k+1}))$ and $a = \mathbf{symbol}(s_{k+1})$ — substitution; $s_{k+1} \neq s_0$

$v \in C_k(\mathbf{prefix}(s_{k+1}))$, $a = \mathbf{symbol}(s_{k+1})$
- $\Rightarrow$ $\mu_k(s_0, v) = \mathbf{prefix}(s_{k+1})$, — by ind. hyp.
- $\Rightarrow$ Case: $\mu_{k+1}(s_0, v) \neq s_{k+1}$
  - $\Rightarrow$ $\mu_{k+1}(s_0, v) = \mu_k(s_0, v) = \mathbf{prefix}(s_{k+1})$ — Obs. 3.1

  Case: $\mu_{k+1}(s_0, v) = s_{k+1}$
  - $\Rightarrow$ $\mu_{k+1}(s_{k+1}, \mathbf{symbol}(s_{k+1})) = s_{k+1}$ — Corollary 3.1
- $\Rightarrow$ $\mu_{k+1}(\mu_{k+1}(s_0, v), \mathbf{symbol}(s_{k+1})) = s_{k+1}$
- $\Rightarrow$ $\mu_{k+1}(s_0, va) = s_{k+1}$ — substitution

$\mu_{k+1}(s_0, va) = s_{k+1}$
- $\Rightarrow$ $\mu_{k+1}(\mu_{k+1}(s_0, v), \mathbf{symbol}(s_{k+1})) = s_{k+1}$
- $\Rightarrow$ Case: $\mu_{k+1}(s_0, v) \neq s_{k+1}$
  - $\Rightarrow$ $\mu_{k+1}(s_0, v) = \mu_k(s_0, v) = \mathbf{prefix}(s_{k+1})$; — Obs. 3.1

  Case: $\mu_{k+1}(s_0, v) = s_{k+1}$
  - $\Rightarrow$ $\mu_{k+1}(s_0, v) \neq \mu_k(s_0, v)$ — $s_{k+1} \notin S_k$
  - $\Rightarrow$ $\mu_k(s_0, v) = \mathbf{suffix}(s_{k+1})$ — Obs. 3.2
  - $\Rightarrow$ $\mu_{k+1}(\mu_{k+1}(s_0, v), a) = \mu_{k+1}(s_{k+1}, \mathbf{symbol}(s_{k+1})) = s_{k+1}$ — substitution
  - $\Rightarrow$ $\mu_k(\mathbf{suffix}(s_{k+1}), \mathbf{symbol}(s_{k+1})) = \mathbf{suffix}(s_{k+1})$ and $\mathbf{suffix}(s_{k+1}) = \mathbf{prefix}(s_{k+1})$ — Lemma 3.2
- $\Rightarrow$ $\mu_k(s_0, v) = \mathbf{prefix}(s_{k+1})$ — substitution.
- $\Rightarrow$ $v \in C_k(\mathbf{prefix}(s_{k+1}))$ and $a = \mathbf{symbol}(s_{k+1})$ — by ind. hyp.□

4.1.2 Proof that for $i \neq k+1$, $va \in C_{k+1}(s_i) \Leftrightarrow \mu_{k+1}(s_0, va) = s_i$

Here, the contrapositive of the result proved in Section 4.1.1, above, shall be useful:

$$va \notin C_{k+1}(s_{k+1}) \quad \Leftrightarrow \quad \mu_{k+1}(s_0, va) = s_i \text{ for some } i \neq k+1. \tag{11}$$

Furthermore, the fact that the languages of disjoint subtrees are disjoint is required. That is,

Claim 4.1

$$va \in C_{k+1}(s_i),\ i \neq k+1,\ \textit{and}\ s_i \neq \mathbf{suffix}(s_{k+1}) \quad \Rightarrow \quad va \notin \mathbf{L}(s_{k+1})$$

Proof: *We proceed by a proof by contradiction.*

$s_i \neq \mathbf{suffix}(s_{k+1})$ *and* $va \in C_{k+1}(s_i)$
$\Rightarrow$ $C_{k+1}(s_i) = C_k(s_i)$ *and* $va \in C_k(s_i)$ $\quad \mathbf{extns}_{k+1}(s_i) = \mathbf{extns}_k(s_i)$
$\Rightarrow$ $\mu_k(s_0, va) = s_i$ *by ind. hyp.*
$va \in \mathbf{L}(s_{k+1})$
$\Rightarrow$ $va \in C_k(\mathbf{prefix}(s_{k+1})) \cdot \mathbf{symbol}(s_{k+1})$ *by def.* $\mathbf{L}()$
$\Rightarrow$ $\mu_k(s_0, v) = \mathbf{prefix}(s_{k+1})$ *by ind. hyp.*
$\Rightarrow$ $\mu_k(s_0, va) = \mathbf{suffix}(s_{k+1})$ *by def.* $\mathbf{suffix}()(\rightarrow\leftarrow)$

The remainder of the proof of Theorem 4.1 follows easily:

$va \in C_{k+1}(s_i), i \neq k+1$
$\Leftrightarrow$ $va \in \left(\mathbf{L}(s_i) - \bigcup_{s_d \in \mathbf{extns}_{k+1}(s_i)} \mathbf{L}(s_d)\right)$ by def. $C_{k+1}()$
$\Leftrightarrow$ $va \in \left(\mathbf{L}(s_i) - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} \mathbf{L}(s_d)\right) - \mathbf{L}(s_{k+1})$ $s_i \neq \mathbf{suffix}(s_{k+1})$ and Claim 4.1 or $s_i = \mathbf{suffix}(s_{k+1})$, therefore $s_{k+1} \in \mathbf{extns}_k(s_i)$
$\Leftrightarrow$ $va \in \left(\mathbf{L}(s_i) - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} \mathbf{L}(s_d)\right) - \left(\mathbf{L}(s_{k+1}) - \bigcup_{s_d \in \mathbf{extns}_{k+1}(s_{k+1})} \mathbf{L}(s_d)\right)$ $\mathbf{extns}_{k+1}(s_{k+1}) = \{\}$
$\Leftrightarrow$ $va \in C_k(s_i) - C_{k+1}(s_{k+1})$ by def. $C_k()$
$\Leftrightarrow$ $va \in C_k(s_i)$ and $va \notin C_{k+1}(s_{k+1})$
$\Leftrightarrow$ $\mu_k(s_0, va) = s_i$ and $\mu_{k+1}(s_0, va) \neq s_{k+1}$ by ind. hyp. and (11)
$\Leftrightarrow$ $\mu_{k+1}(s_0, va) = s_i$ $i \neq k+1$, Obs. 3.1 $\square$

5 DMC vs. Other Stochastic Data Models

There are three important aspects of a stochastic data modeling technique that may distinguish it abstractly from others: the language family of the models constructed, which ultimately determines the limitations of the technique; model structure, which determines how the model organizes the data it gathers; and the statistical assumptions, which determine how the frequency data is updated and combined to compute probability estimates.

These are not orthogonal issues. Usually the frequency data are organized with respect to conditioning contexts, which correspond to partitions on the set of possible strings. Since a model's structure certainly determines the languages its states recognize, a structural point of view is, for the most part, a different (and more intuitive) way of looking at the generative power of the model. But only for the most part: two models that recognize the same languages can have very different structure, and therefore may organize frequency data differently.

5.1 Linguistic Power

Given that the partition element that corresponds to an arbitrary DMC model state cannot be described using a single finite suffix, what, generally speaking, does it take to describe it? There is a known family of languages that contains the family of languages recognizable by DMC FSMs more tightly than the class of regular languages: For any DMC model state s, $C_k(s)$ is a *locally testable star-free* regular set. *Star-free*

regular languages are regular sets that can be described using a finite number of set concatenations, unions, and complements, [16, Chapter 1].

To see that $C_k(s_i)$ is star-free for any s_i, simply replace set subtraction with intersection of the absolute complement of the subtrahend, and A^* with $\overline{\{\}}$. $C_k(s_i)$ is also *locally testable*, which means expressible as finite Boolean combinations of sets of the type FA^*, A^*G, or A^*HA^*, where F, G, H are finite sets. However, this is a loose characterization, for it was shown in [3] the languages recognizable by DMC models are contained, possibly properly, in the class of languages expressible as $A^*F \cup G$, that is, *finite-order* languages. Below, we give a new proof of that containment, which also shows that the characterization of Theorem 4.1, $C_k()$ is *finite-order*, as it should be.

Intuitively, DMC models belong to the class of *finite-order Markov sources* [1], also known as *Finite-Context Automata* [3], because only a finite suffix of a given source string is required to determine the state to which the string will carry a given model. This means, for one thing, that DMC FSMs cannot recognize infinitely repeating patterns, a common capability of more general FSMs. Neither, however, can any of the popular FSMX models [7, 6, 9, 11, 12, 13, 17, 18], all of which can determine the current model state by locating the state whose single associated finite context is a maximal suffix of the given input string. On the other hand, while a given state in an FSMX model can be uniquely specified with a single finite string, an arbitrary DMC state cannot. Therefore, even when stripped of its implementation details and Bayesian assumptions for probability estimation, DMC proves to be distinct from the other stochastic models in the literature.

5.1.1 DMC Models Have Finite Order

The regular expression $C_k(s_i)$ may be described as a *finite-order* characterization of the strings which bring M_k to s_i from the initial state s_0, because deciding membership of any string wa in the language of any state s_i requires a finite number of comparisons with a finite number of symbols at the end of the string wa. The concept, *finite-order*, is easily formalized for regular expressions (and therefore for finite-state machines), and leads to a simple proof that DMC constructs finite-order FSMs.

A language L over a finite alphabet A is *finite order* if and only if there exist finite sets of strings F and G such that $L = A^*F \cup G$.

Finite-Order Expressions over A and the languages they denote are defined recursively:

1. A^* is a finite-order expression and $L(A^*) = A^*$.

2. If α and β are finite-order expressions over A and $a \in A$ then $\alpha \cup \beta$, $\alpha \cap \beta$, $\overline{\alpha}$, and $\alpha \cdot a$ are finite-order expressions, where

$$\begin{aligned} L(\alpha \cap \beta) &= L(\alpha) \cap L(\beta) \\ L(\alpha \cup \beta) &= L(\alpha) \cup L(\beta) \\ L(\overline{\alpha}) &= A^* - L(\alpha) \\ L(\alpha \cdot a) &= L(\alpha) \cdot a. \end{aligned}$$

Theorem 5.1 *A language L is finite-order iff $L = L(\alpha)$, for some finite-order expression α.*

Proof: To show that any finite order language can be expressed as a finite-order expression, we consider a finite-order language $L = A^*F \cup G$, where $F = \{x_1, \ldots, x_n\}$, and $G = \{y_1, \ldots, y_m\}$, and then express the sets A^*F, and G, as finite-order expressions:

$$\begin{aligned} A^*F &= A^*x_1 \cup A^*x_2 \cup \cdots \cup A^*x_n, \quad \text{and} \\ G &= \{\lambda\}y_1 \cup \{\lambda\}y_2 \cup \cdots \cup \{\lambda\}y_m \\ &= \overline{A^*A}y_1 \cup \overline{A^*A}y_2 \cup \cdots \cup \overline{A^*A}y_m. \end{aligned}$$

To prove the contrapositive, we show, by induction on expression length, that the language of any finite-order expression can be expressed as a finite order language, $A^*F \cup G$, where F and G are finite. The basis is trivial: $A^* = A^*\{\lambda\} \cup \{\}$. For the step, consider a finite-order expression that is composed of two shorter finite-order expressions α and β, which by induction hypothesis have corresponding finite sets $F_\alpha = \{x_1, \ldots, x_n\}$, $F_\beta = \{y_1, \ldots, y_m\}$, $G_\alpha = \{v_1, \ldots, v_r\}$, and $G_\beta = \{w_1, \ldots, w_s\}$. There are four cases:

$$\begin{aligned} L(\alpha \cdot a) &= A^*(F_\alpha \cdot \{a\}) \ \cup \ G_\alpha \cdot \{a\}. \\ L(\alpha \cup \beta) &= A^*(F_\alpha \cup F_\beta) \ \cup \ (G_\alpha \cup G_\beta). \\ L(\alpha \cap \beta) &= A^*F \cup G, \\ &\quad \text{where } G = \bigcup_{\substack{1 \le i \le n \\ 1 \le k \le r}} (A^*x_i \cap v_k) \cup \bigcup_{\substack{1 \le j \le m \\ 1 \le l \le s}} (A^*y_j \cap w_l) \cup \bigcup_{\substack{1 \le k \le r \\ 1 \le l \le s}} (v_k \cap w_l), \\ &\quad \text{and } F = \{x_i \in F_\alpha : \exists y_j \in F_\beta \text{ s.t. } y_j \text{ is a suffix of } x_i\} \cup \\ &\qquad \{y_j \in F_\beta : \exists x_i \in F_\alpha \text{ s.t. } x_i \text{ is a suffix of } y_j\}. \\ &\quad \text{The definition of } F \text{ follows from the fact that} \\ &\qquad A^*F = \bigcup_{\substack{1 \le i \le n \\ 1 \le j \le m}} (A^*x_i \cap A^*y_j). \\ L(\overline{\alpha}) &= \overline{A^*x_1 \cup A^*x_2 \cup \cdots \cup A^*x_n \cup v_1 \cup v_2 \cup \cdots \cup v_p} \\ &= \overline{A^*x_1} \cap \overline{A^*x_w} \cap \cdots \cap \overline{A^*x_n} \cap \overline{v_1} \cap \overline{v_2} \cap \cdots \cap \overline{v_p}, \\ &\quad \text{where } \overline{v_i} = A^*Av_i \cup \overline{A^*v_i}, \\ &\qquad \overline{A^*x} = \{\}, \text{ if } x = \lambda; \\ &\qquad \overline{A^*x} = \overline{A^*a_1a_2\cdots a_z}, \text{ if } x = a_1a_2\cdots a_z \\ &\qquad\quad = \{\lambda\} \cup A^*\overline{a_z} \cup A^*\overline{a_{z-1}}a_z \cup \cdots \cup A^*\overline{a_1}a_2\cdots a_z, \\ &\qquad\quad \overline{a} = \{b \in A : b \neq a\}. \end{aligned}$$

Thus, any finite order expression $\overline{\alpha}$ can be transformed into a finite-order expression constructed from single symbols and A^* using only concatenation, intersection, and union; that is, an expression for which the induction is already proved□

The fact that DMC models are finite-order may now be expressed clearly:

Corollary 5.1 *The language of a state s_i in a DMC model M_k, $C_k(s_i)$, is finite-order for all i and k.*

Proof: *The recursive definition of $C_k(s_i)$ reduces to a finite-order expression, that is,*

$$C_k(s_i) \;=\; \mathbf{L}(s_i) \cap \bigcap_{s_d \in \mathbf{extns}_k(s_i)} \overline{\mathbf{L}(s_d)}.$$

5.1.2 DMC Models Are Not FSMX

It is instructive to consider restricted DMC embodiments in which $\mathcal{E}()$ allows transitions to be redirected only once and disallows the redirection of transitions from states that already have extensions. In such models, Equation (10) holds for every state. In that case,

$$C_k(s_i) = A^*\mathbf{context}(s_i) - \bigcup_{s_d \in \mathbf{extns}_k(s_i)} A^*\mathbf{context}(s_d),$$

which is logically equivalent to the closed-form transition function of FSMX models.

FSMX models, defined by Rissanen [12], are characterized by their state-set and transition function:

- The states satisfy a *suffix property*, where states mapped to unique finite strings, or conditioning contexts, and for every state with context x, there exist states for each proper suffix of x. The set of maximal conditioning contexts that have states in the model is determined by a growth heuristic, and varies among different techniques.
- The transition function has a closed form based on state conditioning contexts. The next state is always the state whose associated finite context is the maximal suffix, relative to the context partition defined by the model, of the already-processed portion of the input sequence.

DMC transition functions cannot, in general, be expressed in such a simple closed form.

When $\mathcal{E}()$ allows transitions to be redirected more than once, distinct states s_i and s_j may be mapped by the function **context**() to the same string:

$$\begin{aligned}\mathbf{context}(s_i) = \mathbf{context}(s_j) \;\Rightarrow\; &\exists\; l < |\mathbf{context}(s_i)| \text{ s.t.} \\ &\mathbf{prefix}^l(s_i) = \mathbf{prefix}^l(s_j),\end{aligned}$$

where $\mathbf{prefix}^l()$ is the function obtained by composing the function **prefix**() with itself $l-1$ times.

Furthermore, a completely independent complication arises when transitions from a state that already has extensions may be redirected. In this case, the state corresponding to $M_k(wa)$ will not always be the state in M_k whose **context**() is the longest match to the end of wa. An example of this situation occurs with respect to an arbitrary state s_i in an arbitrary model M_k, where:

$$\begin{aligned}\mathbf{context}(s_i) &= \text{'abc'}, \\ \mathbf{context}(\mathbf{prefix}(s_i)) &= \text{'ab'}, \\ \mathbf{context}(s_d) &= \text{'wabc' for some } s_d \in \mathbf{extns}_k(s_i), \\ \mathbf{context}(\mathbf{prefix}(s_d)) &= \text{'wab', and} \\ \mathbf{context}(s_r) &= \text{'vwab' for some } s_r \in \mathbf{extns}_{d-1}(\mathbf{prefix}(s_d)).\end{aligned}$$

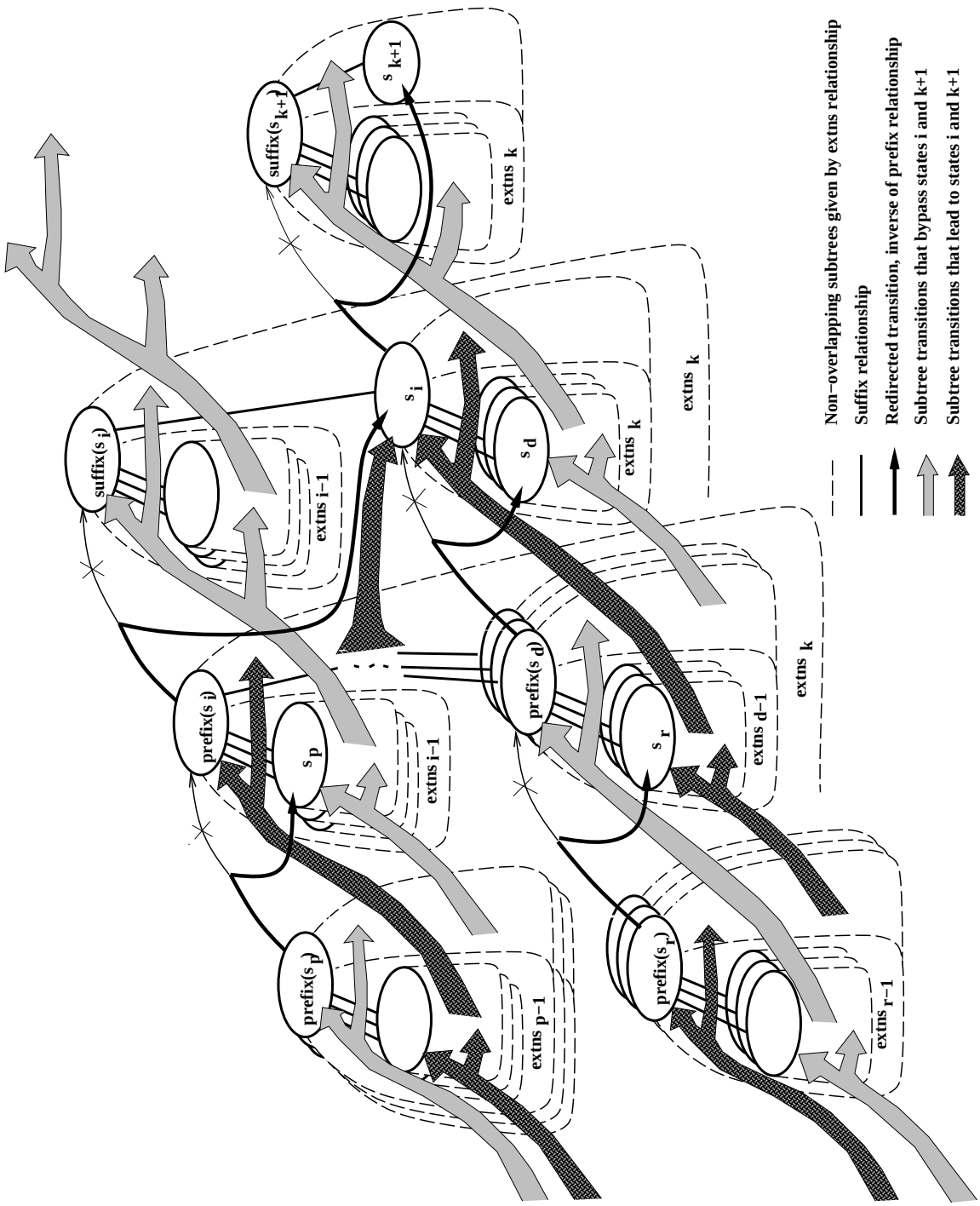

Figure 3: **Example-independent illustration** of the (necessary) alternating recursion of function $\mathbf{L}()$. The states in the upper subtrees correspond to contexts in the first term of $C_k()$, which describes strings that bring M_k to the subtree headed by s_i. The states in the lower subtrees correspond to contexts in the second term of $C_k()$, which describes strings that have been directed away from s_i into s_i's subtree. Note, the figure does not illustrate multiply-redirected transitions.

(Figure 3 is included for use as a template for correctly visualizing particular examples such as this one by writing given contexts on the states with the above labels.) Here, **context**(s_d) is the best-matching context for any input substring in ending in 'vwabc'; however, substrings ending in 'vwabc' take M_k to s_i, not s_d. This example proves that DMC states cannot be uniquely characterized with single strings.

Thus a stronger characterization is required for DMC models than for FSMX models, because DMC models can recognize languages for which no FSMX model exists. Furthermore, for any FSMX model there exists a linguistically equivalent DMC model. Another interesting consequence of the fact that DMC is not FSMX is that this proves conclusively that the family of languages generated by FSMX models does not properly contain the languages generated by Markov Models. This apparently contradicts conventional wisdom that FSMX models "generalize the class of Markov Models" [13].

5.2 Structural Comparison

The **context**() of a DMC state is analogous to the finite contexts of the states in FSMX models [7, 6, 9, 11, 12, 13, 17, 18] vis à vis the terms of the definition of $C_k()$, and the fact that every string that takes a DMC model to a given state s_i ends in **context**(s_i), which is proved in Lemma 3.1. Both DMC and FSMX models are tree-structured, where children states correspond to minimal extensions of their parent's context. Thus DMC and FSMX models are similar enough to allow meaningful comparison of their abstract structural differences, in terms of conditioning contexts.

Since FSMX models are characterized by a uniformly restricted version of DMC's characterizing function $C_k()$, we know that any FSMX model can be simulated by a DMC model. However, such a simulation will require extra states in the DMC model, for the class of DMC models does not contain the class of FSMX models (or *vice versa*). DMC models are strictly Markovian, that is, the next state is always a function of the current state and the currently scanned symbol; whereas the next state in an FSMX model may also depend upon which states were visited before the current state. Note that it is equivalent to say that DMC models satisfy a *prefix* property: for every state s there exist states p for each proper prefix of **context**(s) such that $\mu_k(p, w) = s$, where **context**(s) = **context**(p) $\cdot\, w$. A DMC model that simulates an FSMX model, a Markovian FSMX model, would satisfy both the prefix property above, and the suffix property satisfied by FSMX models. The comparison between such a model and an arbitrary DMC model explores structural differences besides the Markov (or prefix) property.

The first difference is that a DMC model can have distinct states whose associated conditioning context partition elements have the same maximum common suffix. The function **context**() describes these maximum common suffixes. FSMX models also associate conditioning context partition elements with each state, but each is fully characterized by its maximum common suffix, and therefore there is no duplication of the maximum common suffix of any state. The duplication in DMC is partially caused by redirecting the original transition for which a given state was created (that state's *prefix transition*). Any benefit of redirecting a prefix transition is realized only if the other transitions entering the given state, or earlier crossings of the prefix transition itself, have caused the frequency distribution on the state's outgoing transitions (described in Section 5.3) to become contaminated (i.e., non-representative). The cost of redirecting prefix transitions in the absence of such contamination is pervasive model redundancy. Not only can this hinder structural convergence, but the

convergence of statistical parameters may be slowed because message statistics are distributed unnecessarily.

A more important distinction is DMC's capacity for variable-length minimal extensions of a context. This capacity alone sets DMC's (Markovian) data model apart from all FSMX models. Techniques that build FSMX models [7, 6, 9, 11, 12, 13, 17, 18] all grow models with *single-character* minimal extensions. Thus DMC has the capacity for modeling the most probable substrings in a given sequence using fewer states than any of its counterparts.[5]

The remaining structural difference is due to the fact that the (temporal) order in which finite substrings in a given sequence become recognizably frequent strongly affects the structure of the regular sets of conditioning contexts associated with each state. Thus the structure of DMC models reflects higher-order statistics of the conditioning contexts themselves. In contrast, the structure of FSMX models only reflects the zero-order statistics of the conditioning contexts (i.e., how many times each context appears in the source message). Therefore, DMC's Markov models record information about the input sequence that FSMX models do not.

5.3 Bayesian Assumptions and Local Order Estimation

The remaining undescribed aspect of DMC is how it computes probability estimates. As each symbol is processed, DMC simply produces a frequency distribution over A, from the frequency data at the current state, and then increments the symbol count which corresponds to the current symbol at that state. The key to the computation of the estimate is which values are assumed for the recorded frequency of each symbol in A given the current state, when it is first added to the model. For when a new state is first added, the *observed* frequencies of all symbols given that state are zero, and therefore cannot be the basis for a probability estimate. Thus, DMC computes a *prior frequency distribution* for each newly added state, when it is created.

This contrasts sharply with the practice of two other families of on-line algorithms. Rissanen's *Context* algorithm and variants [9, 11, 12, 13, 17] typically assume the same uniform prior for each state as it is added to the model. The assumed prior is therefore not based upon any frequency data. Such simplistic prior assumptions incur negligible penalties later in long target sequences; they are often combined with information-theoretic state-selection that dynamically (re)estimates the local order of the model, and which requires that frequency counts at all excited states be incremented for each symbol. At the other extreme, the PPM algorithm and subsequent variants [7, 6, 18] compute a frequency distribution that is a dynamically weighted average of the distributions at all excited states (i.e., states whose conditioning contexts match the current input history). This "blending" technique is algebraically equivalent to assuming a (pseudo-Bayesian) prior frequency distribution that is (re)computed at the latest possible moment, and which therefore employs information that was not available when the new state was first added to the model. With blending, frequency updates are best made only at certain of the highest-order excited states. DMC's (Bayesian) priors are based only upon frequency data that have been gathered up to the moment that a given state was added, and frequencies are updated only at the highest-order excited state.

[5] That is, assuming the sequence was generated by a Markov FSMX source. Using a Markov model such as DMC's to simulate any non-Markovian FSMX source requires systematic addition of extra states.

5.3.1 Frequency Distributions in DMC

In DMC's intial model, M_0, a uniform frequency distribution is assumed over the reflexive out-transitions of s_0. As each s_{k+1} is added, it becomes the new destination of a transition, $\mathbf{edge}(s_p, c)$, on some c from some s_p, and each out-transition from the original destination, s_t, is copied and assigned a frequency $f_{k+1}()$. That is,

$$\forall b \in A,\ f_{k+1}(\mathbf{edge}(s_{k+1}, b)) = \mathbf{ratio} \cdot f_k(\mathbf{edge}(s_t, b)),$$

where $s_t = \mu_k(s_p, c)$ and

$$\mathbf{ratio} = f_k(\mathbf{edge}(s_p, c))/\Sigma_{b \in A} f_{k+1}(\mathbf{edge}(s_t, b)).$$

Then, the frequencies of the copied out-transitions are *reduced* by the frequencies assigned to the copies,

$$\forall b \in A,\ f_{k+1}(\mathbf{edge}(s_t, b)) = f_k(\mathbf{edge}(s_t, b)) - f_{k+1}(\mathbf{edge}(s_{k+1}, b)).$$

For all other transitions, $f_{k+1} = f_k$.

This process is pictured in Figure 4, using the the definitions of Section 3.1, that is, $s_t = \mathbf{suffix}(s_{k+1})$ and $s_p = \mathbf{prefix}(s_{k+1})$. We call the resulting frequency distribution on the edges leaving s_{k+1} a *retraction*, since the process of redirecting an edge to a new state can be viewed as a reclassification of the strings which take M_k across that edge. Note the resulting 'homogeneity assumption' regarding the distribution of next-symbol frequencies that are conditioned by the redirected edge, relative to the original frequency distribution at its original destination $\mathbf{suffix}(s_{k+1})$ that is implicit here. A more exact (and expensive) way to implement retractions would require having each edge remember the exact distribution of next-symbol frequencies that are conditioned by strings which have crossed the edge in the past.

When encoding an input sequence, DMC's model simply computes the frequency, relative to the sum of the frequencies of the transitions leaving the current state s_i, of the transition leaving s_i that corresponds to the currently scanned symbol c. The resulting conditional probability estimate may then be encoded in as few as $-\log(p_{\text{est}}(c|s_i))$ bits.

This does not directly affect DMC's structure, of interest here. But, we note that the prior distribution assumption that DMC invokes by redistributing frequencies between a new state s_{k+1} and $\mathbf{suffix}(s_{k+1})$ salvages DMC's compression performance, because it hedges against the gross *local order over-estimation* induced by the definition of $\mathcal{E}()$ of [8] described earlier. *Order estimation* is performed in all FSMX models, at least implicitly, to decide how long the contexts represented by the model should be. *Local order estimation* decides whether individual contexts in the model should be shorter or longer. Since $C_k(s_i)$ is minimal and $A^*C_k(s_i) \subseteq A^*\mathbf{context}(s_i)$, the order of any state s_i in a DMC model is at least $|\mathbf{context}(s_i)|$.

The following are immediate corollaries of our main result:

1. DMC states have a defined, locally-variable minimum order.

2. Every state has the same or higher minimum order than its parent.

3. The act of invoking the criterion $\mathcal{E}()$ to decide when to add a state s_{k+1} as an extension to a given s_t is indeed local order estimation.

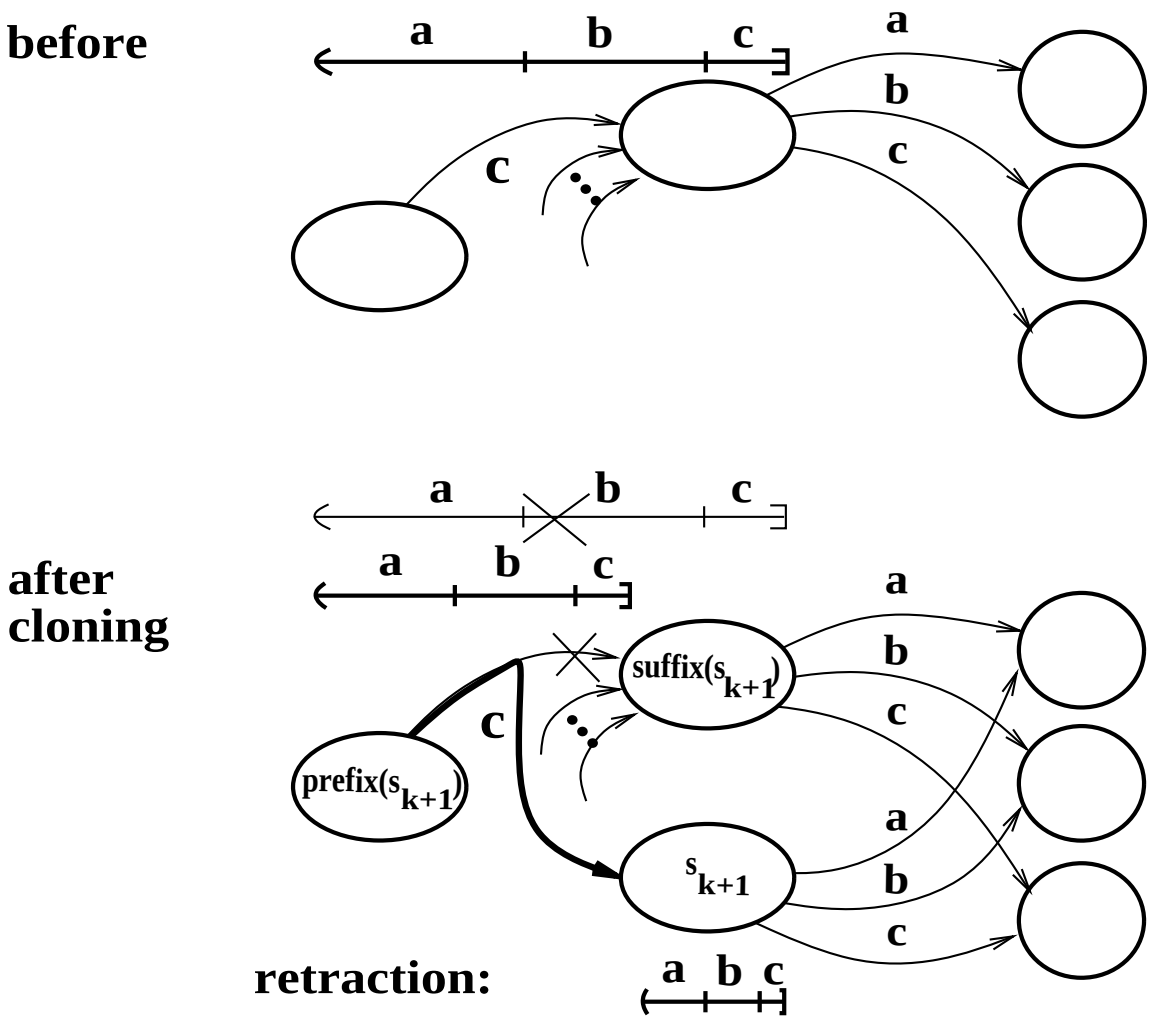

Figure 4: **Cloning, Frequency Distributions, and Retractions in DMC.** When a selected edge redirected, its new destination is a clone of its former destination, and the frequency distribution at the original destination is uniformly divided between the two destinations. The frequencies in the clone's copy of the distribution (the *retraction*) sum to the number of times the redirected edge has been traversed, and the original destination state keeps the remaining frequencies. Note that the number of distribution subinterval calculations and the number of newly added edge pointers corresponds to the size of the input alphabet $|A|$, so in practice, DMC has always taken a binary input alphabet.

6 Curbing Counterproductive Model Growth

An obvious way to reduce the number of states in DMC models is to use a larger input alphabet. However, as originally presented, DMC is only feasible with a binary alphabet, since $|A|$ out-transitions are created for every new state. Other authors [14, 15, 19] have independently generalized DMC to larger alphabets using variations of what we call *lazy cloning*, which copies out-transitions only as needed. However, only the solutions in [4, 14] successfully reduced DMC's memory requirements without eliminating DMC's advantages. The others all had a similar approach: when the required out-transition was absent from the current state, the needed transition was copied from a state that was essentially a zero- or first-order state.

Our analysis implies that the best state from which to copy out-transitions one at a time is the same state they would have been copied from if the copying were done all at once: the **suffix**() state. This way, the transitions go to a next state with the longest possible matching conditioning context. Copying out-transitions from a low-order state has the opposite effect. Thus our analysis of DMC explains the successes and failures of the various practical approaches.

Several natural solutions for curbing DMC's counterproductive model growth follow from our characterization of DMC. They apply individually and in combination.

1. $\mathcal{E}()$ should better approximate entropy-based local order estimation. The frequency of a particular edge does say something about the probability of the state it leads to. Thus the original $\mathcal{E}()$ crudely approximates the contribution of that destination state to the entropy of the model. The other approximation extreme is exemplified by Rissanen [13] and Furlan [9]. These authors closely approximate the relative entropies between states and their minimal extension states using counters which keep track of the code-length differences. The middle ground, that is, effective but quick and imprecise approximations, is utterly unexplored.

2. The refinement eligibility criterion $\mathcal{E}()$ should prevent transitions from being redirected more than once.[6]

3. The transitions exiting any given state should be copied from its **suffix**() one at a time, as they are needed, see Figure 5. When a state does not have the required out-transition, the probability estimate can be made from its **suffix**(). This way, larger input alphabets can be economically accommodated, since only conditioning contexts which have been seen before will be represented by the model. Probability estimation can proceed using the recursive blending technique of PPM [7], or by lazy evaluation of DMC's prior distribution assumption [4]. For example, if blending method PPMC [10] were used, the frequency on the **suffix**() edge should always equal the number of distinct symbols which have been seen when in a given state, and the application of such optimizations as *exclusions* and *update exclusions* [2] is straightforward. Lazy evaluation of DMC's prior distribution assumption, that is, *lazy retractions*, is described in Figure 5.

[6] *Caveat emptor*: J. Teuhola and T. Raita report [personal correspondence] unsatisfactory results when this restriction is combined with lazy cloning. In [4] we investigate whether the success of this restriction depends upon the nature of the refinement criterion $\mathcal{E}()$, and whether transition frequencies are incremented before or after new transition creation.

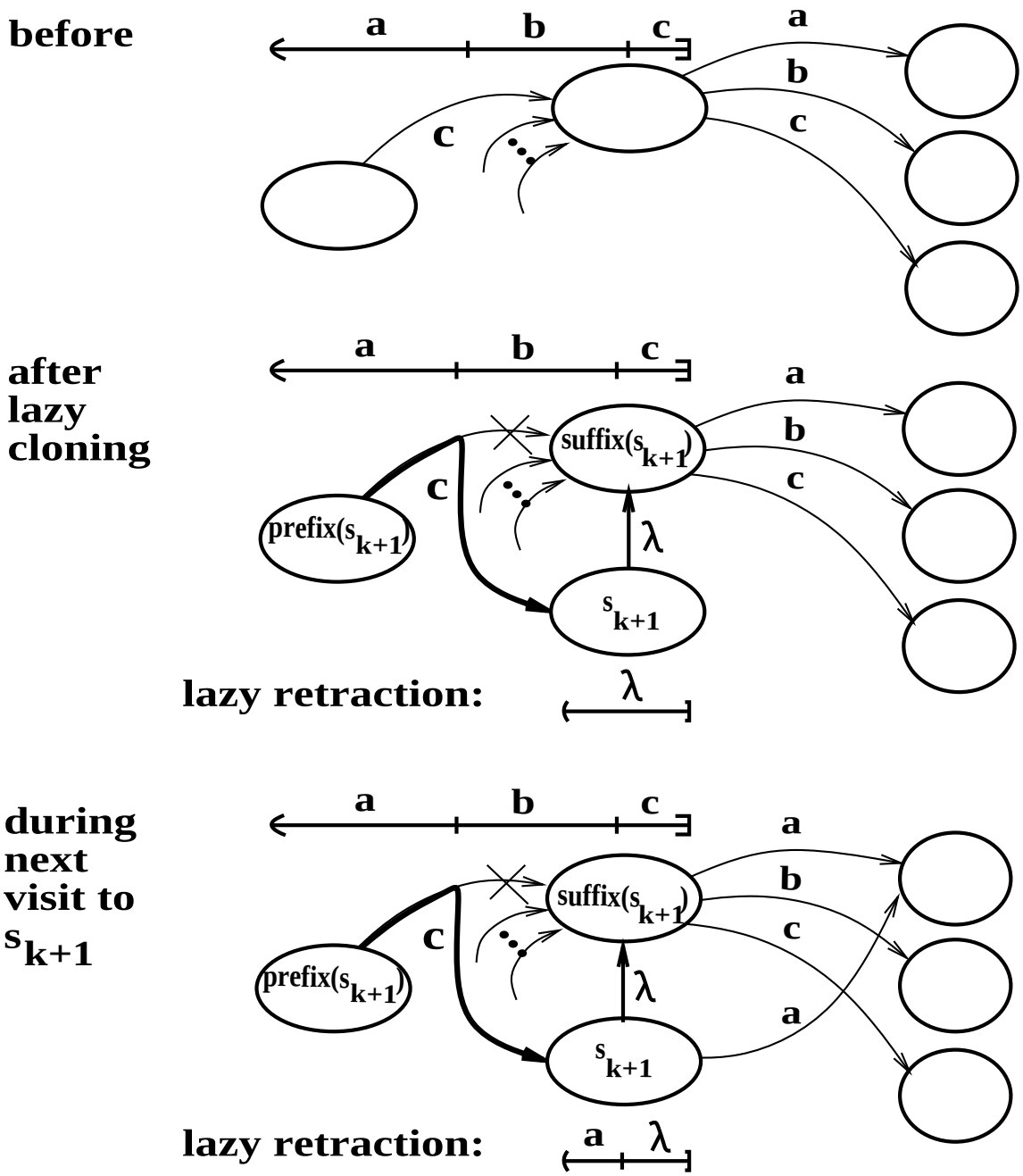

Figure 5: **Lazy Cloning and Retractions in DMC.** When a new state is created, the frequency of the event that a novel symbol is seen (λ) is set to the frequency of the redirected edge, and thus the probability of traversing the **suffix**() pointer when in that state is 1.0. Whenever a novel symbol is seen in a given state, its portion of the state's retraction is recursively evaluated, before the symbol's probability is estimated and before any frequency updates occur at the given state. Note that a novel symbol's portion of the retraction is subtracted from λ's frequency and added to the symbol's frequency. Frequency updates are discussed in detail in [4].

The analysis in this paper holds for all of the DMC variations outlined above. Some of the lazy cloning ideas outlined in item 3 above and explored in [4] were independently developed in [14]. This study of DMC contributes to a larger body of work [4] that examines the combination of convergent local order estimation, aggressive model growth, and statistically aggressive pseudo-Bayesian assumptions in FSMX models and DMC variants.

7 Acknowledgements

The author is extremely grateful to Richard Ladner, for carefully reading the proofs in a previously published version of this paper [5], which contained numerous errors that the author had become blind to during its many revisions. Dr. Ladner also pointed out the clarifying effect of adding a formal proof that DMC models are finite-order, and provided a sketch of the proof of Theorem 5.1.

References

[1] R. Ash. *Information Theory.* John Wiley and Sons, New York, New York, 1965.

[2] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression.* Advanced Reference Series. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[3] T. C. Bell and A. Moffat. A note on the DMC data compression scheme. *The Computer Journal*, 32(1):16–20, 1989.

[4] S. Bunton. *On-Line Stochastic Processes in Data Compression.* PhD thesis, University of Washington, Aug. 1995. (in preparation).

[5] S. Bunton. The structure of DMC. In *Proceedings Data Compression Conference.* IEEE Computer Society Press, Mar. 1995.

[6] J. G. Cleary, W. J. Teahan, and I. H. Witten. Unbounded length contexts for PPM. In *Proceedings Data Compression Conference.* IEEE Computer Society Press, Mar. 1995.

[7] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[8] G. V. Cormack and R. N. S. Horspool. Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550, 1987.

[9] G. Furlan. An enhancement to universal modeling algorithm context for real-time applications to image compression. In *IEEE Transactions on Acoustics Speech and Signal Processing*, pages 2777–2780, 1991.

[10] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[11] T. V. Ramabadran and D. L. Cohn. An adaptive algorithm for the compression of computer data. *IEEE Transactions on Communications*, 37(4):317–324, 1989.

[12] J. J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory*, 29(5):656–664, 1983.

[13] J. J. Rissanen. Complexity of strings in the class of Markov sources. *IEEE Transactions on Information Theory*, 32(4):526–532, 1986.

[14] J. Teuhola and T. Raita. Application of a finite-state model to text compression. *The Computer Journal*, 36(7):607–614, 1993.

[15] T. Y. Tong. *Data Compression with Arithmetic Coding and Source Modeling.* PhD thesis, University of Waterloo, 1993.

[16] J. van Leeuwen. *Handbook of Theoretical Computer Science, Volume B, Formal Models and Semantics.* Advanced Reference Series. The MIT Press, 55 Hayward Street, Cambridge MA 02142, 1990.

[17] M. J. Weinberger, A. Lempel, and J. Ziv. A sequential algorithm for the universal coding of finite memory sources. *IEEE Transactions on Information Theory*, 38(3):1002–1014, 1992.

[18] R. N. Williams. *Adaptive Data Compression.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[19] T. L. Yu. Hybrid dynamic Markov modeling. In *Proceedings Data Compression Conference.* IEEE Computer Society Press, Mar. 1993.