Exact Regular-Constrained Sampling for Variable-Order Markov Generation

François Pachet
LIP6, Sorbonne Université
pachet@gmail.com

Abstract

We study how to sample a sequence of a fixed length from a variable-order Markov model while satisfying regular constraints, such as ending with a specified symbol, placing given symbols at selected positions, or avoiding copied fragments. The model predicts each next symbol from a context of preceding symbols and backs off to a shorter context when needed. Existing belief-propagation methods solve this problem for first-order Markov chains, which use only the last symbol as context. Our contribution is an exact extension to variable-order models. We build a graph of the contexts observed in the training data and combine it with a finite automaton that checks the constraints. This retains the context needed by the predictor and yields samples from a fixed finite-memory source conditioned on the constraints. The backward pass takes time proportional to the requested sequence length times the number of reachable transitions in this combined graph. The method also supports data augmentation through reversible transformations without storing the transformed training sequences, while preserving the probabilities obtained by learning from all the copies. Exhaustive enumeration verifies the constrained probabilities, and Bach Prelude experiments demonstrate practical generation with combined musical constraints.

1 Introduction

Many controlled sequence-generation tasks ask for two properties at once. The sample should be locally plausible under a learned style model, and it should also satisfy explicit finite-horizon requirements. In music, the first property is often supplied by variable-order Markov models: the generator conditions on the longest suffix of the history with observed continuations, backing off when the high-order context is absent, sparse, or too deterministic [Pearce and Wiggins, 2004].

We consider the class of regular constraints: requirements that a finite automaton can check as it reads the generated symbols. This class includes fixed symbols or allowed sets at selected positions, prescribed endings, finite metrical patterns, forbidden substrings, and combinations of these constraints [Pesant, 2004, Roy and Pachet, 2013, Papadopoulos et al., 2014]. The MAXORDER anti-copy constraint is one instance. With parameter M, it forbids every contiguous M-symbol fragment that occurs in the training corpus, so a generated sequence can copy at most $M - 1$ consecutive symbols.

Small creative corpora also motivate data augmentation: applying reversible transformations to the training sequences to obtain additional examples. In music, these transformations include pitch transpositions and register shifts. Storing every transformed copy would multiply the stored corpus. Instead, we compute each source row—the possible next symbols, their probabilities, and the next contexts—by looking up counts in the original corpus after applying the inverse transformations. BP uses these rows just as it would use rows learned from explicit transformed copies.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Markov constraints and later belief-propagation (BP) methods solve this problem for finite first-order chains [Pachet and Roy, 2011, Papadopoulos et al., 2015, Roy and Pachet, 2013]. Equivalently, once the correct stochastic finite-state source is known, one can compose it with a regular acceptor and run a dynamic program on the product [Mohri et al., 2002, Allauzen et al., 2007]. We therefore do not claim a new automata-composition algorithm. The missing point is source identification: which finite-state object correctly represents a variable-order/backoff generator? A solver whose state is only the last emitted symbol is exact for a first-order projection, but not for the variable-order process whose probabilities depend on longer observed suffixes. Related model-checking and constrained-decoding work also starts from a fixed labeled transition system or decoder and applies automata-style restrictions [Baier and Katoen, 2008, Hokamp and Liu, 2017, Post and Vilar, 2018]. Our focus is the source identification and sparse compilation step for count-based variable-order generators.

For computing conditional probabilities, we consider the current context as the state of the variable-order source: the longest stored suffix of the symbols already generated. This state determines the next-symbol probabilities and the context update. For example, contexts $(1, 2)$ and $(6, 2)$ can predict different continuations even though both end in 2. We pair this context with the state of the constraint automaton. The latter records the information needed to check the regular constraint. For MAXORDER, it tracks whether recent symbols match the beginning of a forbidden corpus fragment. BP then computes how much accepted probability remains after each candidate symbol. Suffix trees or context trees can store the contexts, and WFSA/WFST software can run the product inference; our construction specifies the source states and probabilities that must be supplied to those tools.

We compare four approaches to computing constrained continuations. A first-order projection uses only the last symbol to compute future probabilities. Dense order-K lifting stores every length-K suffix, allowing $|\mathcal{V}|^K$ states. The observed-context approach uses the contexts stored by the trained model. An order stack tests several context lengths during generation and chooses one that permits a valid completion. Table 1 compares their sampling guarantees and graph sizes before we propose a graph construction.

Concretely, we formalize the conditional distribution induced by a fixed variable-order/backoff source under a regular constraint, give a finite counterexample showing why first-order feasibility and future-mass heuristics can produce the wrong posterior, and run BP on the reachable product of the sparse context graph and the constraint automaton. The resulting construction has preprocessing cost $O(n|E_{\otimes}|)$ for horizon n and reachable product edges $E_{\otimes}$. Enumeration tests verify exactness, while a Bach Prelude experiment shows that the sparse product remains practical under a regular anti-copy constraint. We distinguish exact conditioning of a fixed source from policies that select a context length during generation, such as choosing the longest context that permits a valid completion or skipping a context with only one continuation. Reversible augmentation is handled by the same compiled source without expanding the stored corpus.

2 Problem setup

Let $\mathcal{V}$ be a finite alphabet. A trained variable-order model stores continuation counts $N(c, y)$ for contexts c of length at most K. Given a history $h_t = (x_0, \ldots, x_{t-1})$, an order policy selects a suffix c_t of h_t, typically the longest suffix with usable continuations, and samples

$$P_{\mathrm{VO}}(x_t = y \mid h_t) = \frac{N(c_t, y)}{\sum_{y'} N(c_t, y')}$$

or a smoothed variant computed from the same context state. The unconstrained sequence probability is

$$P_{\mathrm{VO}}(x_0, \ldots, x_{n-1}) = \prod_{t=0}^{n-1} P_{\mathrm{VO}}(x_t \mid x_0, \ldots, x_{t-1}).$$

A regular constraint is defined by a deterministic finite automaton $A = (Q, \mathcal{V}, \delta, q_0, F)$. A generated sequence is valid when the run from q_0 ends in F. Positional constraints are the special case in which the automaton state records the position; MAXORDER or anti-plagiarism constraints can be represented by automata that reject when a forbidden training fragment has just been completed [Papadopoulos et al., 2014]. The target distribution, when $Z_{\mathrm{VO}}(A) > 0$, is

$$P_{\mathrm{VO}}(x \mid x \in L(A)) = \frac{1}{Z_{\mathrm{VO}}(A)} P_{\mathrm{VO}}(x) \mathbf{1}[x \in L(A)]. \tag{1}$$

The normalizing constant $Z_{\mathrm{VO}}(A)$, also called the partition function, is the total probability that the source assigns to valid length-n sequences. The challenge is that x_t alone is not a Markov state for this source.

The phrase "variable order" can refer either to model estimation or to generation-time behavior. In estimation, the classical question is which context tree should be kept from data. Rissanen's context algorithm selects a variable-length context tree by a minimum-description-length criterion [Rissanen, 1983]; related variable-length Markov-chain methods use penalized likelihood, BIC-like pruning, or statistical tests [Bühlmann and Wyner, 1999]. Ron, Singer, and Tishby's probabilistic automata with variable memory length and later variable-order Markov prediction work give closely related finite-memory source models [Ron et al., 1996, Begleiter et al., 2004]. Context-tree weighting instead mixes over many trees and has a universal coding interpretation [Willems et al., 1995]. These methods decide which suffixes are statistically useful for prediction, or how to combine their predictions. They do not by themselves answer the finite-horizon control question: once a source or order policy has been fixed, how should that source be conditioned on a regular language?

There is also a classical data-structure lineage behind the sparse contexts we use. Suffix trees and related string indexes compactly represent observed suffixes and substrings, with linear-time constructions going back to Weiner, McCreight, and Ukkonen, and with standard treatments such as Gusfield's monograph [Weiner, 1973, McCreight, 1976, Ukkonen, 1995, Gusfield, 1997]. This literature explains how to store and traverse large sets of observed contexts efficiently. It does not address the probabilistic conditioning problem studied here: exact sampling from a variable-memory stochastic source subject to a finite-horizon regular constraint.

An exact construction can store every length-K suffix as a separate state, giving a first-order chain with $|\mathcal{V}|^K$ possible suffix states. Composing that chain with the regular acceptor permits exact conditioning. A trained context tree usually stores only a small fraction of these K-tuples. The construction below uses those stored contexts as its source states.

Observation 1. *Exact regular-constrained sampling for a variable-order model requires the inference state to contain enough information to determine both the next variable-order predictive distribution and the current state of the regular constraint automaton.*

3 Why first-order conditioning fails

The following finite example illustrates how first-order conditioning changes the probabilities. Let $\mathcal{V} = \{0, 1, 2, 3, 4, 5, 6\}$, maximum order $K = 2$, fixed prefix $(0, 1)$, and training multiset

$$\begin{aligned} 10 &\times [0, 1, 2, 4], \\ 10 &\times [0, 1, 3, 5], \\ 1 &\times [0, 1, 3, 4], \\ 1000 &\times [6, 2, 5], \\ 1000 &\times [6, 3, 4]. \end{aligned}$$

We generate two symbols x_0, x_1 and impose $x_1 = 4$. From context $(0, 1)$, the local variable-order probabilities are

$$P_{\mathrm{VO}}(2 \mid 0, 1) = 10/21, \qquad P_{\mathrm{VO}}(3 \mid 0, 1) = 11/21.$$

Both candidates can reach the required ending, but with different probabilities:

$$P_{\mathrm{VO}}(4 \mid 1, 2) = 1, \qquad P_{\mathrm{VO}}(4 \mid 1, 3) = 1/11.$$

Therefore the exact constrained first-symbol probabilities are

$$P_{\mathrm{VO}}(x_0 = 2 \mid x_1 = 4) = 10/11, \qquad P_{\mathrm{VO}}(x_0 = 3 \mid x_1 = 4) = 1/11.$$

A support-only mask is already wrong because it keeps both candidates and samples from $10/21, 11/21$. Weighting by first-order future mass is worse in this example. The order-1 projection merges histories ending in 2 and in 3, giving

$$P_1(4 \mid 2) = 1/101, \qquad P_1(4 \mid 3) = 1001/1011,$$

which reverses the relevant future preference. The failure is not due to regular constraints; it is due to computing future mass in the wrong state space. Quantitatively, the support-only posterior has total variation distance 0.433 from the exact posterior, while the first-order-future-mass hybrid has total variation distance about 0.900 and chooses the wrong MAP symbol. Thus producing only valid sequences does not ensure the requested conditional distribution: here a valid-sequence sampler can even favor the wrong first symbol.

4 Reference points

The preceding example shows how the inference state affects the constrained probabilities. We compare the following approaches by the context they retain, their sampling guarantees, and their graph sizes.

Support-guided first-order masks. One can run first-order BP on the order-1 projection and use it only as a feasibility mask around the variable-order sampler. This is the engineering strategy used in some interactive systems: remove candidates whose projected future mass is zero, then sample the remaining candidates from the selected variable-order context. It is useful for avoiding immediate dead ends, but it does not compute the posterior in Equation (1). In the example above, both 2 and 3 are feasible, so support alone cannot distinguish $10/11$ from $1/11$.

Weighted first-order hybrids. A more tempting fix is to weight a variable-order proposal by first-order future mass,

$$P_{\mathrm{VO}}(y \mid c_t)\, \beta^{(1)}_{t+1}(y),$$

where $\beta^{(1)}$ is computed on the order-1 projection. This uses future mass rather than support alone, but it is still the future mass of the wrong source. In the integer example it prefers 3, because order 1 has merged the contexts $(1, 3)$ and $(6, 3)$. The heuristic can therefore be worse than a plain support mask.

Dense order-K lifting. At the opposite extreme, dense lifting makes the Markov state the full suffix $(x_{t-K+1}, \ldots, x_t)$. This recovers exactness for a maximum-order source and is a useful oracle for tiny problems, but its state space grows as $|\mathcal{V}|^K$, before multiplying by the acceptor. For musical examples with moderate pitch alphabets and $K = 6$, this is already millions to hundreds of millions of states. Sparse context-state BP keeps the exact future-mass semantics while using only represented contexts and reachable product edges.

State kept	Sampling guarantee	Graph size	Requirement or effect
Last symbol or order-1 projection	Exact only for the projected first-order source	$\lvert\mathcal{V}\rvert$ source states before acceptor product	Forgets symbols preceding the last
Dense order-K suffix	Exact for a fixed order-K source	$\lvert\mathcal{V}\rvert^K$ source states before acceptor product	Stores all length-K suffixes
Observed context graph	Exact for a fixed finite-memory VO/backoff source	$\lvert\mathcal{T}\rvert$ source states and reachable product edges	Each context fixes its next-symbol probabilities
Order stack with BP future mass	Exact BP weights within each selected order	One sparse product per candidate order	Order selection specifies a sequential generation policy

Table 1: Sampling guarantees and graph sizes for the four state choices. The observed-context graph supports exact conditioning of a fixed source; the order stack specifies how to choose an order during generation.

5 Sparse context-state BP

We propose to replace the dense order-K state space by the observed contexts used by the generator.

Definition 1 (Context graph)**.** *Let $\mathcal{T}$ be the suffix-closed set of contexts stored by the variable-order model, including the empty context and any start/end sentinels. For context $s \in \mathcal{T}$ and emitted symbol y, define $\mathrm{canon}(s, y)$ as the longest suffix of $s \cdot y$ that belongs to $\mathcal{T}$. The context graph has an edge $s \xrightarrow{y,p} s'$ when $s' = \mathrm{canon}(s, y)$ and the fixed source assigns probability $p = P(y \mid s) > 0$.*

In implementation, contexts are interned in a trie or hash table with suffix links/failure links. During graph compilation, $\mathrm{canon}(s, y)$ is precomputed by following suffix links from $s \cdot y$ to the longest stored suffix, so BP uses integer state identifiers and transition lists. The index affects only compilation constants; the recurrence below depends only on the compiled finite graph.

This graph is the sparse finite-state source traversed implicitly by the variable-order sampler. It can represent maximum-likelihood longest-suffix models, smoothed suffix mixtures, and any fixed finite-memory policy whose transition probabilities depend only on the represented context. For example, an interpolated suffix model may precompute

$$P_{\text{int}}(y \mid s) = \lambda(s)\widehat{P}(y \mid s) + (1 - \lambda(s))P_{\text{int}}(y \mid \text{suffix}(s)),$$

and use this value as the edge weight. Extra state is needed only when the generation policy has memory not contained in s, such as adaptive counters or mode switches.

For ordinary longest-suffix backoff, source compilation tests the current context and successively drops its oldest symbol until it finds usable continuation counts. It normalizes those counts and stores the probabilities on the outgoing edges. After emitting y, the next state is $s' = \text{canon}(s, y)$, the longest stored suffix of $s \cdot y$. The constrained sampler can then read the probabilities and next states directly from the graph.

Algorithm 3 in Appendix B gives the compilation procedure for a fixed backoff rule. It also accepts pruned context trees and smoothed suffix mixtures: each state must determine the next-symbol probabilities and the next state. An order policy that inspects constrained future probabilities is specified separately below.

The same interface supports reversible data augmentation without copying the training set. Let $\mathcal{G}$ be a finite set of invertible transformations of the alphabet, extended symbolwise to contexts. Instead of inserting all transformed sequences, the row used by Algorithm 3 can be computed by inverse lookup,

$$N_{\mathcal{G}}(s, y) = \sum_{g \in \mathcal{G}} N(g^{-1}s, g^{-1}y), \qquad P_{\mathcal{G}}(y \mid s) = \frac{N_{\mathcal{G}}(s, y)}{\sum_z N_{\mathcal{G}}(s, z)}.$$

For music, $\mathcal{G}$ could be the 12 pitch transpositions or another family of reversible pitch/rhythm transforms. This gives the same source probabilities as a materialized augmented corpus, while the original count table remains the only stored training object; transformed rows are generated lazily, cached, or represented by an original context plus an augmentation tag. The regular BP recurrence is unchanged and still sums over the same augmented paths. Counts from all transformations are pooled at each prediction step; the method does not choose one transposition and retain it for the whole sequence. To retain such a choice, the transformation identity must be included in the source state. This does not by itself quotient the constraint product, but it exposes where transformed product rows repeat. If the constraint language is invariant under the same transformations, contexts can further be quotiented by orbit representatives; if constraints mention absolute symbols, such as ending on C, the emitted labels remain absolute while source probabilities are still computed by orbit aggregation.

Proposition 1 (Finite-state sufficiency)**.** *Let G be any finite stochastic source with labeled edges $s \xrightarrow{y,p} s'$ and initial state s_0, such that every emitted sequence has probability equal to the product of edge weights along its unique source path. For any deterministic acceptor A, backward-weighted sampling on the reachable product $G \otimes A$ draws exactly from the source conditioned on $L(A)$, whenever its acceptance probability is positive.*

The proof is the usual automaton-product dynamic program: product paths carry source probability $\prod_t p_t$ and terminal weight $\mathbf{1}[x \in L(A)]$, so backward messages compute the remaining accepted mass and the normalized edge scores are exactly the conditional chain rule. The role of Algorithm 3 is to provide the correct sparse source G for a fixed variable-order/backoff generator.

For positional constraints with potentials $\psi_t(y)$, the backward messages over contexts are

$$\beta_t(s) = \sum_{s \xrightarrow{y,p} s'} p\, \psi_{t+1}(y)\, \beta_{t+1}(s'), \qquad \beta_n(s) = 1.$$

The locally correct constrained sampling score from state s_t is

$$S_t(y, s') = p(s_t, y, s')\, \psi_{t+1}(y)\, \beta_{t+1}(s').$$

Proposition 2. *If the context graph edge probabilities are the transition probabilities of the fixed variable-order source, backward-weighted ancestral sampling on the context graph samples exactly from the source conditioned on the positional constraints.*

The proof is the ordinary forward-backward proof applied to the context graph: each emitted sequence corresponds to one path whose weight is exactly $P_{\text{VO}}(x)$, and the potentials multiply that path weight by the constraint indicator or soft weight. A full proof is in Appendix A.

Algorithm 1 Regular-constrained context-BP sampler

Require: context graph $G = (\mathcal{T}, E_{\mathcal{T}})$, acceptor $A = (Q, \mathcal{V}, \delta, q_0, F)$, horizon n, initial context s_0

build reachable product edges $(s, q) \xrightarrow{y,p} (s', \delta(q, y))$
set $\beta_n(s, q) \leftarrow \mathbf{1}[q \in F]$
for $t = n-1, n-2, \ldots, 0$ **do**
    for all reachable product states (s, q) **do**
        $\beta_t(s, q) \leftarrow \sum_{(s,q) \xrightarrow{y,p} (s',q')} p\, \beta_{t+1}(s', q')$
    end for
end for
if $\beta_0(s_0, q_0) = 0$ **then report** no accepted continuation and stop
$s \leftarrow s_0$, $q \leftarrow q_0$, $x \leftarrow ()$
for $t = 0, \ldots, n-1$ **do**
    sample an outgoing edge $(s, q) \xrightarrow{y,p} (s', q')$ proportionally to $p\, \beta_{t+1}(s', q')$
    append y to x; $s \leftarrow s'$; $q \leftarrow q'$
end for
return x

6 Regular constraints

For a general deterministic acceptor $A = (Q, \mathcal{V}, \delta, q_0, F)$, the BP state is the product

$$(s, q) \in \mathcal{T} \times Q.$$

For every context edge $s \xrightarrow{y,p} s'$ and defined automaton transition $q' = \delta(q, y)$, add the product edge

$$(s, q) \xrightarrow{y,p} (s', q').$$

Determinism gives each generated sequence one automaton run, so its probability is counted once. An ambiguous nondeterministic automaton can have several accepting runs for the same sequence; it must first be determinized for this construction. The backward recursion is

$$\beta_t(s, q) = \sum_{(s,q) \xrightarrow{y,p} (s',q')} p\, \beta_{t+1}(s', q'), \qquad \beta_n(s, q) = \mathbf{1}[q \in F].$$

Sampling then chooses each outgoing product edge proportionally to $p\, \beta_{t+1}(s', q')$.

Proposition 3. *This sampler draws exactly from $P_{\mathrm{VO}}(x \mid x \in L(A))$, provided the context graph edge probabilities are those of the fixed source being conditioned.*

For a corpus of N events, each position contributes at most $K + 1$ observed context-continuation pairs, including the empty context. Hence the number of distinct observed contexts and count entries is $O(N(K + 1))$, rather than $|\mathcal{V}|^K$. Backoff or smoothing may add outgoing transitions; we measure inference cost using the edges actually compiled.

The cost is $O(n|E_{\otimes}|)$, where $E_{\otimes}$ is the reachable product edge set for horizon n. Since $|E_{\otimes}| \leq |Q|\,|E_{\mathcal{T}}|$, the worst case is the standard automaton product, but the implementation builds only reachable states and edges. Memory is $O(n|V_{\otimes}|)$ for a sparse backward table, or one layer when only the partition function is required. A larger alphabet can give each context more outgoing symbols, and a more complex acceptor can make more context-acceptor pairs reachable. Sparse source storage therefore does not guarantee a small product for every constraint.

For repeated sampling with the same source, prefix, horizon, and constraints, the backward table is computed once. Each draw looks up the outgoing weights at the current context–acceptor state. Caching their cumulative sums also avoids rebuilding the local sampling distribution on each visit.

Generation-time order policies. An order-selection policy may choose at each step the longest context with positive backward mass, or skip a context with one feasible continuation. Such a policy defines its own sequence probabilities. In the experiments, we report the probability that a fixed source satisfies the constraints (its partition function), and the probability that the stated policy generates the full sequence without failure. Exact conditioning of a policy-defined source requires its order decisions to be included in the source state and probabilities.

For an order stack, let G_k keep contexts of length at most k, and let $\beta^{(k)}$ be its backward table for the requested horizon and constraints. At time t, let s_k be the longest suffix of the generated prefix h_t stored in G_k. Each outgoing edge receives weight

$$W_k(y, s') = P_k(y \mid s_k)\, \psi_{t+1}(y)\, \beta^{(k)}_{t+1}(s', q'), \qquad q' = \delta(q, y).$$

Here $\psi_{t+1}(y)$ is a positional mask, set to 1 when no positional restriction applies. Algorithm 2 tries the orders from highest to lowest and samples at the first order with positive total weight. For example, a context may have observed continuations whose backward weights are all zero because none can reach the required ending. The algorithm then tries the next shorter context. Singleton avoidance can add a further test before accepting an order; the policy must state that test explicitly.

Algorithm 2 One longest-feasible backoff step

Require: generated prefix h_t, position t, acceptor state q, graphs $G_1, \ldots, G_K$, backward tables $\beta^{(k)}$, masks ψ

for $k = K, K-1, \ldots, 1$ **do**
  $s_k \leftarrow$ longest suffix of h_t stored in G_k
  $C_k \leftarrow \emptyset$
  for all edges $s_k \xrightarrow{y,p} s'$ with defined $q' = \delta(q, y)$ **do**
    $w \leftarrow p\, \psi_{t+1}(y)\, \beta^{(k)}_{t+1}(s', q')$
    if $w > 0$ **then** add (y, s', q', w) to C_k
  end for
  if $C_k \neq \emptyset$ **then**
    sample (y, s', q', w) from C_k proportionally to w
    return y, k, s', q'
  end if
end for
return failure

7 Experiments

We test whether the proposed sampler reproduces the constrained probabilities obtained by enumerating short sequences, and whether the graph of reachable context–automaton states supports interactive sampling on a larger musical example. The augmentation test compares probabilities and storage when transformed training sequences are stored explicitly or accessed by inverse lookup. All backward messages are computed exactly, without pruning accepted paths. Reproducibility details are in Appendix D.

7.1 Exactness by enumeration

For short horizons, we compute

$$Z_{\mathrm{brute}}(A) = \sum_{x \in \mathcal{V}^n} P_{\mathrm{VO}}(x) \mathbf{1}[x \in L(A)]$$

by enumeration and compare it with $Z_{\mathrm{BP}} = \beta_0(s_0, q_0)$. We compare the conditional probabilities from enumeration and BP, and the frequencies in 20000 samples. The first test uses the two-symbol ending constraint from Section 3. The second uses an order-2 source over $\{0, 1, 2, 3\}$, prefix $(0, 1)$, and four generated symbols, with consecutive symbols $(2, 2)$ forbidden. Its automaton has two accepting states: no trailing 2, or one trailing 2. Reading a second consecutive 2 enters a rejecting state. Table 2 shows individual sequence probabilities; Table 3 compares the full distributions. BP matches enumeration up to floating-point precision, and all samples satisfy the constraints. Appendix C gives the training sequences.

Sequence	$P_{\mathrm{VO}}(x)$	Exact / BP cond.	Sample freq.
Required second symbol: $x_1 = 4$			
$(2, 4)$	0.4761904762	0.9090909091	0.9108
$(3, 4)$	0.0476190476	0.0909090909	0.0892
Avoid consecutive symbols $(2, 2)$*: five largest probabilities*			
$(2, 3, 0, 1)$	0.1282051282	0.1872246696	0.1912
$(0, 3, 1, 2)$	0.0961538462	0.1404185022	0.14145
$(3, 0, 1, 2)$	0.0854700855	0.1248164464	0.1214
$(0, 2, 1, 3)$	0.0769230769	0.1123348018	0.11315
$(3, 2, 0, 1)$	0.0672268908	0.0981749528	0.09775

Table 2: Conditional probabilities from enumeration and BP, with frequencies from 20000 samples. The two calculations agree to the displayed precision. The regular-constraint test has 13 accepted sequences with positive source probability.

Metric	Positional $x_1 = 4$	Avoid $(2, 2)$
Z_{brute}	0.52380952381	0.684766214178
Z_{BP}	0.52380952381	0.684766214178
$\lvert Z_{\mathrm{brute}} - Z_{\mathrm{BP}} \rvert$	0	0
$\mathrm{TV}(\text{exact, BP})$	0	4.60×10^{-17}
$\mathrm{TV}(\text{exact, empirical})$	0.00170909	0.00924597
Sample violations	0	0
Context states / edges	11 / 23	18 / 40
Reachable product states / edges	4 / 4	19 / 36

Table 3: Enumeration checks for positional and regular constraints. Empirical TV uses 20000 samples.

7.2 Bach scalability

The larger experiment uses a pitch-only encoding of Bach's Prelude in C. The corpus has 592 pitch events and an alphabet of 25 MIDI pitches. We use horizon $n = 32$, maximum source order $K \in \{1, \ldots, 6\}$, a final pitch-class constraint requiring the last event to be C, and a MAXORDER constraint with $M = 5$: no generated pitch 5-gram may occur in the training corpus. The MAXORDER acceptor has 657 states; the final-C constraint is applied as a positional mask rather than multiplied into the acceptor. Count rows are unsmoothed maximum-likelihood continuation distributions; other fixed smoothing or interpolation choices would change only the source edge weights, not the product-BP construction.

The main run uses a longest-feasible constrained order stack. For each maximum order K, graphs for orders $1, \ldots, K$ are prepared. At each step the policy scans high to low, skips orders with zero BP future mass, selects the highest feasible order, and samples within that order proportionally to local edge probability times next backward mass. In this setting the policy success mass is 1 for every K, and every sample satisfies both constraints.

K	$\lvert\mathcal{T}\rvert$	$\lvert E_{\mathcal{T}}\rvert$	Prod. states	BP edges	$\lvert\mathcal{V}\rvert^K$	BP / sample
1	25	117	606	74115	25	0.0756s / 0.230ms
2	167	449	929	82395	625	0.0908s / 0.310ms
3	524	1080	1122	83162	15625	0.0896s / 0.349ms
4	1180	2062	1161	83195	390625	0.0902s / 0.374ms
5	2187	3437	1175	83195	9765625	0.0902s / 0.368ms
6	3587	5246	1205	83207	244140625	0.0896s / 0.385ms

Table 4: Bach constrained order-stack experiment. Product states count distinct context-acceptor pairs separately for each order. BP edges count expansions over all time steps and orders. Times are medians over five measured repeats; sample time is per sequence.

Table 4 shows the central scaling behavior. At $K = 6$, the order stack visits 1205 context-acceptor states, counted separately per order, and expands 83207 time-indexed edges. Dense order-K lifting

would have $25^6 = 244140625$ states. The measured backward pass remains below 0.10 seconds and sampling below 0.4 ms per generated sequence.

n	$\sum_t \lvert V_{\otimes,t}\rvert$	$\sum_t \lvert E_{\otimes,t}\rvert$	BP (s)	Sample (ms)	Peak MiB
16	10128	33546	0.127	0.037	2.4
32	24833	83651	0.452	0.090	9.1
64	54273	183875	1.208	0.189	24.7
96	83713	284099	2.072	0.295	40.3
128	113153	384323	2.834	0.380	53.6

Table 5: Horizon ablation for Bach constrained order-stack BP with $K = 6$ and MAXORDER $M = 5$. Counts are time-indexed dynamic-program states and edges. All rows have policy success mass 1, zero constraint violations, and maximum copied span 4.

Table 5 varies the horizon while holding the source order and MAXORDER strength fixed. Time-indexed product edges grow roughly linearly over this range, the backward pass remains below 3 seconds at $n = 128$, and amortized sampling remains below 0.4 ms per sequence after the backward table is built. The ablation measures peak Python allocation with `tracemalloc` enabled during BP. Its times include that instrumentation and should not be compared directly with the uninstrumented times in Table 4.

K	Fixed-source mass	Policy success mass	Violations	Max copied span
1	6.955×10^{-3}	1	0	4
2	3.212×10^{-9}	1	0	4
3	0	1	0	4
4	0	1	0	4
5	0	1	0	4
6	0	1	0	4

Table 6: Bach diagnostic separating exact conditioning of a fixed longest-observed-suffix source from the constrained order-stack policy. The MAXORDER-5 constraint forbids copied 5-grams, hence the maximum copied span is four.

Table 6 shows why the policy distinction matters. Conditioning a fixed pure longest-observed-suffix source gives tiny mass at $K = 1$, nearly zero mass at $K = 2$, and exact zero mass for $K \geq 3$. The constrained order stack instead has success mass 1 for every K: it uses BP future mass to skip orders whose high-order continuations cannot satisfy the final-C and MAXORDER constraints. In the measured $K = 6$ runs, each repeat samples $100 \times 32 = 3200$ events; between 3106 and 3119 events use order 2, and the remaining 81 to 94 use order 3. Higher orders contribute no decisions because they have zero feasible future mass at the start under MAXORDER-5.

7.3 Virtual augmentation and constrained sample

Exact virtual transposition augmentation. Table 7 compares explicit and inverse-lookup augmentation under the Bach MAXORDER and final-C constraints. Both give the same success probability and initial backward masses for every order. Inverse lookup stores 592 corpus events and visits 896 source states; explicit augmentation stores 7104 events and builds 27371 source states. Both process the same number of BP edges, with similar backward-pass times. Appendix E gives the anchored generation results and the measurements of repeated transition rows under transposition.

Qualitative constrained sample. Figure 1 shows a compact Bach Prelude-style example generated with explicit anchor constraints. The horizon is 64, the maximum source order is $K = 4$, and the policy is longest-feasible constrained backoff. In addition to an anti-copy constraint forbidding generated 6-grams from the training corpus, the run fixes pitches C, D, E, F, G, A, B, C at positions $0, 8, \ldots, 56$. The anchors are satisfied, and the sample has zero anti-copy violations. This is not a separate quality evaluation; it illustrates that the same regular-product machinery can combine a planned pitch trajectory, local Prelude-like arpeggiation, and an active anti-copy language.

Method	Stored events	Full source states	Visited source states	BP edge expansions	BP (s)
Original corpus	592	3587	3587	82491	0.020
Stored copies	7104	27371	27371	1968532	0.468
Inverse lookup	592	27371	896	1968532	0.497

Table 7: Virtual augmentation matches explicit materialized augmentation (0 success-mass difference and matching start-order masses) while storing only the original Bach sequence. BP edges count expansions over time and orders.

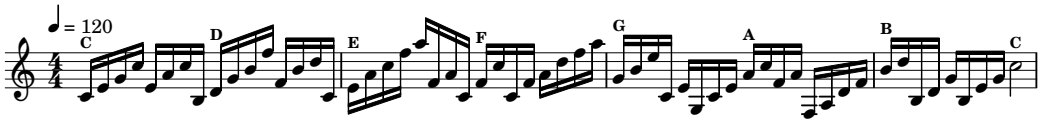


Figure 1: Bach Prelude-style sample with cyclic anchor constraints and a MAXORDER anti-copy constraint.

8 Limitations and impact

Our exactness claim is conditional on the finite-state source whose edge weights are passed to BP. If a generator changes its order policy online, performs singleton avoidance, or adapts its model during generation, that behavior must be folded into the source state or described as a separate sequential policy. The experiments are also deliberately narrow: the exactness checks are tiny because enumeration is required, and the larger study uses one musical corpus and one family of anti-copy constraints. Other corpora, determinization-heavy regular languages, and memory-profile sweeps remain important follow-up tests. Larger alphabets, such as polyphonic encodings or richer symbolic event types, may increase both context branching and acceptor products; we have not yet characterized those regimes. A neural language model would require its predictions and state updates to be represented by a manageable finite set of states before this computation could be applied.

The positive impact is improved controllability and reproducibility for finite-state generative systems, especially in creative domains where explicit constraints and anti-copy controls are desirable. The main risk is misuse of controlled generation systems to imitate protected styles or produce derivative material.

9 Conclusion

This work enables variable-order Markov generators to produce finite sequences that satisfy regular constraints. The method retains the contexts used by the predictor and computes the probability of a valid completion after each possible next symbol. For a fixed finite-memory source, sampling with these probabilities gives exactly the source distribution conditioned on the constraints.

Enumeration tests confirm the conditional probabilities, and the Bach experiments show how backward messages also guide longest-feasible order selection to satisfy both an ending requirement and an anti-copy rule. Data augmentation through reversible transformations reproduces the probabilities obtained by learning from transformed copies, while storing only the original corpus. The resulting method samples exactly from finite-memory variable-order models conditioned on regular constraints, with a backward pass linear in the sequence length and the number of reachable context–automaton transitions.

Acknowledgments and disclosures

This work received no external funding. The author declares no competing interests.

References

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. OpenFst: A General and Efficient Weighted Finite-State Transducer Library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, volume 4783 of *Lecture Notes in Computer Science*, pages 11–23. Springer, 2007. doi: 10.1007/978-3-540-76336-9_3.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

Ron Begleiter, Ran El-Yaniv, and Golan Yona. On Prediction Using Variable Order Markov Models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004. doi: 10.1613/jair.1491.

Peter Bühlmann and Abraham J. Wyner. Variable Length Markov Chains. *The Annals of Statistics*, 27(2):480–513, 1999. URL `https://statistics.berkeley.edu/tech-reports/479`.

Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

Chris Hokamp and Qun Liu. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1535–1546, 2017. doi: 10.18653/v1/P17-1141.

Edward M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM*, 23(2):262–272, 1976. doi: 10.1145/321941.321946.

Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech & Language*, 16(1):69–88, 2002. doi: 10.1006/csla.2001.0184.

François Pachet. vo-regular-bp: Variable-order regular belief propagation. `https://github.com/fpachet/vo-regular-bp`, 2026. GitHub repository, version v0.1.0.

François Pachet and Pierre Roy. Markov constraints: steerable generation of Markov sequences. *Constraints*, 16(2):148–172, 2011. doi: 10.1007/s10601-010-9101-4.

Alexandre Papadopoulos, Pierre Roy, and François Pachet. Avoiding Plagiarism in Markov Sequence Generation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Alexandre Papadopoulos, François Pachet, Pierre Roy, and Jason Sakellariou. Exact Sampling for Regular and Markov Constraints with Belief Propagation. In *Principles and Practice of Constraint Programming*, volume 9255 of *Lecture Notes in Computer Science*, pages 341–350. Springer, 2015.

Marcus Pearce and Geraint Wiggins. Improved Methods for Statistical Modelling of Monophonic Music. *Journal of New Music Research*, 33(4):367–385, 2004.

Gilles Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. In *Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 482–495. Springer, 2004.

Matt Post and David Vilar. Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1314–1324, 2018. doi: 10.18653/v1/N18-1119.

Jorma Rissanen. A Universal Data Compression System. *IEEE Transactions on Information Theory*, 29(5):656–664, 1983. doi: 10.1109/TIT.1983.1056741.

Dana Ron, Yoram Singer, and Naftali Tishby. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning*, 25(2):117–149, 1996.

Pierre Roy and François Pachet. Enforcing Meter in Finite-Length Markov Sequences. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Esko Ukkonen. On-line Construction of Suffix Trees. *Algorithmica*, 14(3):249–260, 1995. doi: 10.1007/BF01206331.

Peter Weiner. Linear Pattern Matching Algorithms. In *14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973. doi: 10.1109/SWAT.1973.13.

Frans M. J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The Context-Tree Weighting Method: Basic Properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995. doi: 10.1109/18.382012.

A Proofs

Proposition 4. *If the context graph edge probabilities are the transition probabilities of the fixed variable-order source, backward-weighted ancestral sampling on the context graph samples exactly from the source conditioned on positional constraints.*

Proof. By construction, the context state contains the memory used by the predictor. For any emitted sequence $x_0, \dots, x_{n-1}$, repeated application of canon gives a unique path in the context graph, and the product of edge weights along that path is the source probability $P_{\mathrm{VO}}(x)$. Multiplying edge t by $\psi_{t+1}(x_t)$ gives the unnormalized constrained path weight. The backward recursion sums all remaining constrained path weights from each context state. Therefore choosing the next edge proportionally to $p\,\psi_{t+1}(y)\,\beta_{t+1}(s')$ is exactly the chain rule for the normalized constrained distribution. □

Proposition 5. *The regular product sampler draws exactly from $P_{\mathrm{VO}}(x \mid x \in L(A))$, provided the context graph edge probabilities are those of the fixed source being conditioned.*

Proof. Each product path records both a source path and the deterministic run of the acceptor on the same emitted labels. The terminal weight $\mathbf{1}[q \in F]$ removes precisely the paths whose labels are not in $L(A)$. Since product edges keep the source weights, the unnormalized weight of an accepting product path is $P_{\mathrm{VO}}(x)\mathbf{1}[x \in L(A)]$. The backward recursion computes the total remaining accepting mass from every product state, so backward-weighted ancestral sampling draws from the normalized accepted-path distribution. Determinism of the acceptor gives a one-to-one correspondence between accepted sequences and accepted product paths. An ϵ-free NFA can be determinized before the same construction. □

B Source compilation

Algorithm 3 Compile a fixed variable-order/backoff source

Require: suffix-closed contexts $\mathcal{T}$, fixed predictor or counts, fixed backoff rule π, canonical suffix map canon
Ensure: stochastic context graph $G = (\mathcal{T}, E_{\mathcal{T}})$
for all $s \in \mathcal{T}$ **do**
  compute the fixed source distribution $D_s(y) = P_\pi(y \mid s)$
  for all $y \in \mathcal{V}$ such that $D_s(y) > 0$ **do**
    $s' \leftarrow \mathrm{canon}(s, y)$
    add edge $s \xrightarrow{y, D_s(y)} s'$ to $E_{\mathcal{T}}$
  end for
end for
return G

C Enumeration inputs and calculations

In the positional example of Section 3, the exact unnormalized constrained scores for the first generated symbol are

$$S(x_0 = 2) = \frac{10}{21}, \qquad S(x_0 = 3) = \frac{11}{21}\frac{1}{11} = \frac{1}{21}.$$

Thus $Z = 11/21$, giving conditional probabilities $10/11$ and $1/11$. The first-order projection instead computes future masses

$$P_1(4 \mid 2) = \frac{10}{10 + 1000} = \frac{1}{101}, \qquad P_1(4 \mid 3) = \frac{1 + 1000}{10 + 1 + 1000} = \frac{1001}{1011}.$$

The projected state has merged contexts $(1, 2)$ with $(6, 2)$, and $(1, 3)$ with $(6, 3)$. This is why first-order future mass assigns almost all probability to $x_0 = 3$, even though the exact variable-order posterior assigns probability $1/11$ to that choice.

The regular-constraint test uses maximum order 2, prefix $(0, 1)$, and four generated symbols over $\{0, 1, 2, 3\}$. Its weighted training corpus is

$$\begin{aligned} 10 &\times [0, 1, 2, 2, 0, 1, 3, 0], \\ 5 &\times [0, 1, 2, 3, 0, 1, 0, 3], \\ 4 &\times [0, 1, 0, 2, 1, 3, 2, 0], \\ 3 &\times [0, 1, 3, 2, 2, 1, 0, 2], \\ 2 &\times [2, 0, 1, 2, 0, 3, 1, 2]. \end{aligned}$$

The acceptor starts in the state for no trailing 2. It switches to the one-trailing-2 state after a 2, returns after any other symbol, and rejects after two consecutive 2s. Enumeration over all 4^4 generated sequences gives 13 accepted sequences with positive source probability.

D Reproducibility details

The public code is `vo-regular-bp` [Pachet, 2026], available at `https://github.com/fpachet/vo-regular-bp`. The implementation line used for the CPU timing tables led to release `v0.1.0` (commit `0911caf`). The horizon-ablation script was added in commit `df16944`; this later snapshot also contains the earlier experiment scripts. These historical versions identify the paper experiments separately from later library changes. The library is MIT-licensed. The Bach pitch file is `data/bach_prelude_c_major_pitches.txt` in that repository.

The exactness experiments use 20000 samples, with seed 0 for the positional test and seed 1 for the forbidden-substring test. The Bach sweep uses $K = 1, \ldots, 6$, horizon 32, MAXORDER 5, 100 samples per K, seed 0, one warmup run, and five measured repeats. Timings were measured on macOS/Darwin 24.6.0 on an Apple M1 Pro with 10 cores and 32 GB RAM, using Python 3.13.11 in a single Python process. The Bach corpus is a public-domain pitch-only encoding of Prelude in C with 592 pitch events and an alphabet of 25 MIDI pitches. The virtual augmentation run uses integer pitch shifts $-6, \ldots, 5$, $K = 4$, horizon 64, fixed bar-start anchors every eight events, and a singleton-avoiding backoff policy.

The experiment scripts are in the library repository. At the historical snapshots their paths are:

- `scripts/eval_tiny_exactness.py`: enumeration and sampling checks.
- `scripts/eval_bach_scalability.py`: the order sweep, with `-source-policy policy_stack`, orders $1, \ldots, 6$, horizon 32, MAXORDER 5, and the sample/repeat settings above.
- `scripts/eval_neurips_ablation.py` at `df16944`: the horizon sweep $16, 32, 64, 96, 128$, with $K = 6$, MAXORDER 5, seed 20260503, 100 samples, one warmup and five measured repeats. Peak memory is measured by `tracemalloc` around BP; it is not total process memory. The sampled sequences are checked directly for the final pitch class and forbidden copied fragments.
- `scripts/eval_virtual_augmentation.py` at the later snapshot `e126dfa`: explicit versus virtual augmentation and constrained distributions; continuation rows and edges are also checked in the test suite.

The recorded CPU timing runs included local implementation changes; their base commit IDs alone do not describe the exact measured working trees.

E Virtual augmentation details

Table 7 compares no augmentation, explicit 12-key materialization, and virtual 12-key augmentation on the Bach MAXORDER + final-C setup. The virtual backend stores the original corpus and transform family, matches the explicit augmented source exactly, and materializes far fewer lazy source states during BP. Product-edge work is unchanged in this generic exact recurrence.

Validation tests compare virtual augmentation against explicit materialization for continuation rows, graph edges, positional order-stack distributions, and regular/MAXORDER distributions.

Table 8 reports the anchored 12-key transposition run discussed in Section 7. All rows use $K = 4$, horizon 64, integer pitch shifts $-6, \ldots, 5$, and anchors $60, 62, 64, 65, 67, 69, 71, 72$ at positions $0, 8, \ldots, 56$. The MAXORDER constraint is checked against the full virtual augmented corpus. These numbers are for the exact virtual source backend, not for a compressed product recurrence.

M	$\lvert\mathcal{T}\rvert$	$\lvert E_{\mathcal{T}}\rvert$	Prod. states	BP edges	Max order	Max copied span
8	2667	5573	31790	2824222	4	7
7	4226	7404	22531	2610565	3	6
6	4765	7886	14394	2245042	3	5
5	4803	7914	8130	1587777	2	4

Table 8: Virtual 12-key transposition augmentation with anchored Bach generation. All rows satisfy all anchors, have success mass 1, and have no copied M-gram violation. Product states are counted separately per order; BP edges are counted over time steps and orders.

Orbit diagnostics confirm that much of the enlarged product is transformed duplicate structure, but not all of it is reusable under the finite shift set and absolute anchors used here. Product states collapse by about $15\times$ modulo transposition, whereas exact row signatures collapse by about $1.5\times$. The implemented row cache is exact but conservative: it reuses DFA-accepted transition rows while applying positional masks exactly, and speeds the $M = 5$ run by $1.37\times$, with smaller gains for $M \geq 6$. We therefore treat deeper transformation-aware product BP as future work requiring stronger equivariance conditions or a more symbolic recurrence.

F Additional implementation notes

The product is built lazily from states reachable under the initial prefix, horizon, and constraint automaton. Pure positional constraints are kept as time-indexed masks rather than multiplied into the regular DFA. Forbidden substring constraints are compiled to a dense integer Aho-Corasick-style DFA when possible, while a generic deterministic acceptor path remains available. Backward values are memoized as $\beta(t, s, q)$, or as $\beta^{(k)}(t, s, q)$ when an order cutoff is part of an order-stack policy. Candidate sets store cumulative weights so repeated samples from the same prepared backend do not rebuild local distributions.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the limited contribution: exact regular conditioning for a fixed finite-memory variable-order source, plus policy-level use of exact future mass. The limitations section explicitly separates that claim from online order policies.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: The limitations and impact section discusses the fixed-source assumption, order policies, narrow empirical scope, and remaining tests.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: The propositions state the finite-memory and deterministic-acceptor assumptions, with proof sketches in the main text and full proofs in Appendix A.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: The paper specifies the toy distributions, constraints, sample counts, seeds, Bach corpus, horizons, MAXORDER values, timing repeats, and compute environment. Appendix D links the public implementation and identifies historical versions and experiment scripts. It also records the limits of the working-tree metadata for the timing runs.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: Appendix D links the public MIT-licensed library and its Bach pitch file, identifies versions and scripts, and gives the settings used for the experiments. The recorded CPU runs included local changes, so the listed base commits do not alone reconstruct the exact timed working trees.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The paper gives the context orders, horizons, constraints, corpus statistics, sampling counts, timing repeats, seed, and compute setup. There is no neural training optimizer.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

   Justification: Exactness results compare BP to enumeration. Timing results are medians over five repeats without error bars or confidence intervals. We report the measured times and do not claim statistical significance.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Appendix D reports the machine, operating system, CPU, RAM, Python version, and timing protocol.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work uses finite-state algorithms and a public-domain musical example, and does not involve human subjects, private data, or high-risk deployed systems.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes].

    Justification: The limitations and impact section discusses controllability and reproducibility benefits, as well as misuse risks for style imitation and derivative generation.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release a high-risk pretrained model, image generator, or scraped dataset. The MAXORDER constraint is discussed as an anti-copy mechanism, not as a complete release safeguard.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes].

    Justification: Prior algorithms and papers are cited. The Bach score is public-domain source material; the pitch file is included in the linked implementation. The vo-regular-bp repository uses the MIT license.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes].

    Justification: The public implementation includes a README, API documentation, examples, tests, and experiment scripts. Appendix D identifies the paper versions and settings. No new dataset or pretrained model is introduced.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or human-subject experiments.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The paper does not involve crowdsourcing or human-subject experiments.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: LLMs are not part of the core method, experiments, or scientific contribution.