

Reproducing Bayesian Model Merging for Stochastic Context Free Grammar Induction

September 2026

Abstract

Stolcke and Omohundro [8] induce stochastic context free grammars by applying merging and chunking to a grammar that initially memorises the training corpus, retaining changes according to a Bayesian posterior. We present an OCaml implementation and evaluate which reported results it reproduces. The default score combines a description length prior with approximate marginal likelihood, fitting production probabilities by the inside outside algorithm and applying a Dirichlet correction to expected counts. An alternative score uses a mean field variational lower bound. At their respective fixed points, both scores consist of the Dirichlet integral of expected production counts plus parse entropy, evaluated at different parameter settings. They agree when all parses of each string have the same production-count vector; otherwise agreement is not guaranteed, with a difference of 1.42 nats on a small ambiguous grammar. Across the eight languages from the original paper and three additional synthetic targets, the induced languages match the targets up to length seven, and every recursive target yields a recursive grammar. The largest target’s result is taken from the committed experiment because our rerun did not finish. The running corpus $\{ab, aabb, aaabbb\}$, with frequencies in the ratio $6 : 3 : 1$, generalises to $a^n b^n$ at 200 samples but retains the memorising grammar at 300 samples, exhibiting the predicted absolute frequency effect. For four targets, every held out string is present in training. Their held out likelihood therefore establishes fit to observed strings, while evidence of generalisation comes from the recovered recursive structure.

1 Introduction

Learning a probabilistic grammar requires estimating both its productions and their probabilities. With the productions fixed, parameter estimation is a continuous optimisation problem addressed by Baker’s inside outside algorithm [1], developed further by Lari and Young [6] as an instance of expectation maximisation [3]. Selecting the productions requires a discrete search whose space grows combinatorially with the number of nonterminals and the lengths of the right hand sides. Likelihood alone favours a grammar that enumerates the training strings, so it does not supply the structural preference needed for this search.

Bayesian model merging [8] combines structural selection and parameter estimation within one objective. The structural prior penalises model size and the marginal likelihood measures fit after integrating over parameter uncertainty. Starting with one production for each distinct training string, the search applies local generalisation operators while the posterior improves. The original paper recovers the eight test grammars that Cook, Rosenfeld and Aronson [2] used for their hill climbing method, and it also infers the palindromes ww^R over $\{a, b\}$, a language that Cook et al. list as beyond the reach of their algorithm.

We implemented the method in OCaml and evaluated its behaviour on synthetic languages. Section 2 introduces stochastic grammars and the Bayesian objective, followed in Section 3 by the grammar representation, parsing, scoring, editing operators and search procedure. Section 4 relates the two evidence approximations, and Section 5 examines how corpus size affects generalisation in the original paper’s running example. Section 6 reports the experiments on

eleven target languages. The numerical results come from running repository commit 279ec24, except for the basic English result in Table 3, which was taken from the committed run because our rerun did not finish.

2 Background

2.1 Stochastic context free grammars

A stochastic context free grammar consists of a context free grammar $G = (\mathcal{N}, \Sigma, P, S)$ and a probability $\theta_{A \rightarrow \gamma}$ for each production. Probabilities sum to one over the productions sharing a left hand side A . The probability of a derivation is the product of its production probabilities, and the probability of a string is the sum over its derivations. For a string $w = w_1 \cdots w_n$ the inside probability $I(A, i, j)$ is the probability that A derives $w_{i+1} \cdots w_j$, so that $P(w) = I(S, 0, n)$, and the outside probability $O(A, i, j)$ is the probability of deriving $w_1 \cdots w_i A w_{j+1} \cdots w_n$ from S . The expected number of uses of a production $A \rightarrow \gamma$ in the derivations of w is

$$\mathbb{E}[n_{A \rightarrow \gamma}] = \frac{1}{P(w)} \sum_{0 \leq i < j \leq n} O(A, i, j) \theta_{A \rightarrow \gamma} I(\gamma, i, j),$$

where $I(\gamma, i, j)$ is the inside probability of the sequence γ over the span. Expectation maximisation alternates between computing these expected counts over the corpus and setting each probability to its count divided by the total count of its left hand side [1, 6, 3].

2.2 Bayesian model merging

Stolcke and Omohundro [8] distinguish the structure M_S of a grammar M from its parameters θ_M . Structural induction seeks the structure with maximal posterior

$$P(M_S | X) \propto P(M_S) \int P(\theta_M | M_S) P(X | M_S, \theta_M) d\theta_M,$$

where X is the corpus. The structural prior $P(M_S)$ is a description length prior that favours grammars which take fewer bits to write down, and the parameter prior $P(\theta_M | M_S)$ is a product of Dirichlet distributions with one factor for each left hand side. The initial grammar contains one production per distinct corpus string, giving maximal likelihood at the cost of a long description. The search then applies two structural operators. Merging replaces two nonterminals by one that has the union of their productions, which generalises the language and usually shortens the description. Chunking replaces a contiguous sequence of symbols by a fresh nonterminal that expands to it, which leaves the language unchanged but can set up a merge that makes the grammar recursive. A chunk usually lowers the posterior initially. Beam search retains intermediate grammars so that a later merge can realise the improvement which would be missed by accepting only immediate gains.

3 Implementation

The implementation consists of an OCaml library and a command line driver. Figure 1 presents the induction procedure, whose stages are described below.

3.1 Representation and data incorporation

The corpus representation stores distinct token sequences with real valued counts. A grammar stores its start symbol and a list of productions, each consisting of a left hand side, a sequence of terminal or nonterminal symbols on the right, and a count. Following Section 3.3 of the original paper, the initial grammar introduces a nonterminal T_a with the single production

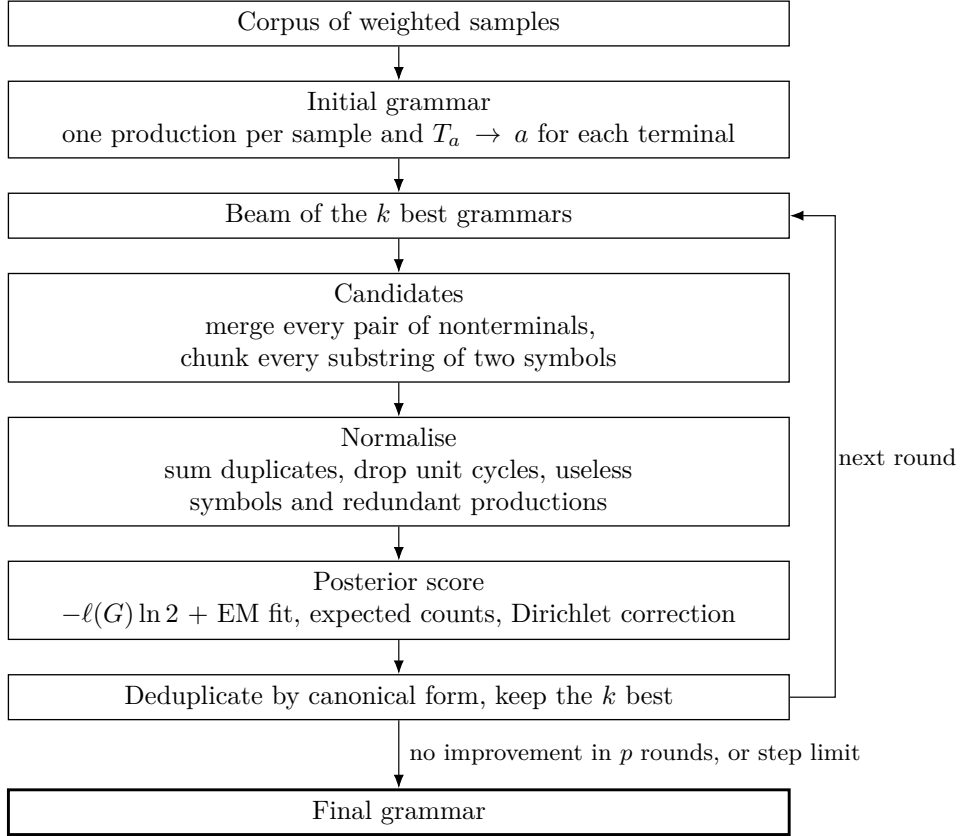


Figure 1: The induction procedure. Each round expands every grammar in the beam by every candidate operator, normalises and scores the results, and keeps the k best distinct grammars, until p rounds pass without improving the best score or the step limit is reached.

$T_a \rightarrow a$ for every terminal a in the corpus, and every distinct training string w with count c_w contributes a production $S \rightarrow T_{w_1} \cdots T_{w_n}$ with that count. Initially, structural productions contain only nonterminals. Merging can therefore operate on terminal classes through their lexical nonterminals using the same operation as for other nonterminals.

3.2 Parsing

The parser computes inside probabilities directly for arbitrary right hand sides, avoiding conversion to Chomsky normal form. It memoises values for nonterminals and right hand side suffixes over spans, enumerating the split between the first symbol of each suffix and the remaining symbols. Empty productions are excluded, so each symbol covers at least one token and the possible splits are bounded. Recursion through productions with multiple symbols reduces the span; unit productions are the exception because they retain its length. The inside pass takes time proportional to n^3 multiplied by the total length of the right hand sides.

The outside computation visits spans from longest to shortest. Within each span, it visits nonterminals in topological order of the unit production graph, completing a parent outside value before propagating it to a child. The parser rejects grammars with cycles in that graph. For each occurrence of a nonterminal in a right hand side the implementation sums over the start and end positions of that occurrence within the parent span, which makes the outside pass as written proportional to n^4 multiplied by the total right hand side length. Both passes store logarithms and combine alternatives with a numerically stable log sum. Expectation maximisation runs for at most eight iterations by default and stops early once the corpus log likelihood changes by less than 10^{-7} .

3.3 Structural prior

The implementation assigns the following description length in bits

$$\ell(G) = \log_2 |\mathcal{N}| + \log_2 |\Sigma| + \sum_{A \in \mathcal{N}} \left[\log_2 |P_A| + \sum_{A \rightarrow \gamma \in P_A} \left(\log_2 |\gamma| + \sum_{x \in \gamma} b(x) \right) \right],$$

where P_A is the set of productions of A , $b(x) = \log_2 |\mathcal{N}|$ for a nonterminal and $b(x) = \log_2 |\Sigma|$ for a terminal, and $\log_2 1 = 0$. The original paper specifies only a cost of $\log_2 |\mathcal{N}|$ bits for each nonterminal occurrence, so the remaining terms, which encode the alphabet sizes, the number of productions of each nonterminal and the length of each right hand side, are implementation choices. The log prior in nats is $-w \ell(G) \ln 2$, where the weight w defaults to one.

3.4 Marginal likelihood

For an unambiguous corpus, parameter integration reduces to a product of closed-form Dirichlet integrals over production counts. Ambiguous strings contribute sums over derivations, so that factorisation does not apply directly. The default approximate evidence score fits $\hat{\theta}$ by expectation maximisation and obtains expected counts $\hat{c}$ under $\hat{\theta}$. It then replaces the maximum likelihood contribution of these counts with their Dirichlet integral. For a left hand side A with k productions, expected counts $c_1, \dots, c_k$ and total c_A , the log Dirichlet integral with symmetric concentration α is

$$\log \text{DM}_A(c) = \log \Gamma(k\alpha) - \log \Gamma(c_A + k\alpha) + \sum_{i=1}^k [\log \Gamma(c_i + \alpha) - \log \Gamma(\alpha)],$$

and the approximate evidence is

$$\mathcal{E}_{\text{approx}}(G) = \sum_{w \in X} c_w \log P(w \mid \hat{\theta}) + \sum_A \left[\log \text{DM}_A(\hat{c}) - \sum_i \hat{c}_i \log \frac{\hat{c}_i}{\hat{c}_A} \right].$$

The concentration α defaults to 0.1. A grammar that leaves any training string without a parse scores $-\infty$, which rules out grammars that fail to cover the corpus.

The alternative score uses a mean field variational lower bound with a Dirichlet distribution over parameters and pseudocounts $\beta_i = \alpha + \mathbb{E}_q[n_i]$, in the style of Kurihara and Sato [4]. The implementation sets the geometric mean weights $\bar{\theta}_i = \exp(\psi(\beta_i) - \psi(B_A))$, where B_A is the sum of the pseudocounts for the left hand side of production i and ψ is the digamma function, runs the inside and outside passes under those weights to obtain expected counts, updates the pseudocounts, and repeats. The bound it reports is

$$\mathcal{E}_{\text{vb}}(G) = \sum_{w \in X} c_w \log Z_w(\bar{\theta}) - \sum_A \text{KL}(\text{Dir}(\beta_A) \parallel \text{Dir}(\alpha)),$$

where $Z_w(\bar{\theta})$ is the inside value of w under the unnormalised weights. A third scoring mode uses maximum likelihood alone, without correcting for parameter uncertainty. The log gamma function uses the Lanczos approximation [5], and the digamma function uses its recurrence together with an asymptotic series.

3.5 Operators and normalisation

Candidate generation considers every unordered pair of nonterminals for merging and every distinct adjacent pair of symbols in a right hand side for chunking. Longer chunks can be enabled through an option. A merge keeps the start symbol if it is one of the pair and otherwise keeps the lexicographically smaller name, so the result does not depend on the order in which the pair was generated. The paper leaves repeated occurrences within one right hand side unspecified. A chunk therefore introduces a fresh nonterminal and, according to an option, replaces either all nonoverlapping occurrences of its sequence or only the leftmost occurrence.

After each edit, normalisation combines duplicate productions and sums their counts. It removes unit productions that close a cycle, then eliminates nonproductive and unreachable nonterminals together with their productions. It also deletes a production if its left hand side can derive the same right hand side as a sentential form without using that production. That last check is a chart computation over the sentential form with unit productions closed at each span, and it removes, for example, the production $S \rightarrow T_a X T_b$ once $S \rightarrow T_a S T_b$ and $X \rightarrow T_a T_b$ are both present.

3.6 Search

The default search retains a beam containing at most k candidate grammars. Each round applies every candidate operator to every grammar in the beam, normalises and scores each result, discards any result that scores $-\infty$, and keeps the k highest scoring distinct grammars as the next beam. The best recorded grammar is updated when the leading candidate improves its score by more than 10^{-9} . Search ends at the round limit or after p consecutive rounds without this improvement. Grammars are deduplicated up to renaming of nonterminals. The implementation decides this by computing a canonical string for each grammar, which it does by refining a colouring of the nonterminals by the multiset of their production signatures and, where a colour class remains ambiguous, branching on each member of the first such class and keeping the lexicographically smallest encoding. This is the individualisation and refinement scheme used for graph canonisation [7], and its worst case is exponential in the size of the ambiguous classes. The driver additionally provides best first search over a bounded frontier, although the experiments reported here use beam search.

4 The two evidence scores

The test suite verifies agreement between the scores of Section 3.4 on an unambiguous grammar. The following proposition expresses both scores at their respective fixed points and identifies a sufficient condition for agreement.

Proposition 1. *Let $H(r)$ denote the entropy of a distribution r over the parses of the corpus, weighted by the string counts. At a fixed point of expectation maximisation, where $\hat{\theta}_i = \hat{c}_i / \hat{c}_A$,*

$$\mathcal{E}_{\text{approx}}(G) = \sum_A \log \text{DM}_A(\hat{c}) + H(r_{\hat{\theta}}),$$

where $r_{\hat{\theta}}$ is the posterior over parses under $\hat{\theta}$. At a fixed point of the variational iteration,

$$\mathcal{E}_{\text{vb}}(G) = \sum_A \log \text{DM}_A(\mathbb{E}_q[n]) + H(q),$$

where q is the posterior over parses under the weights $\bar{\theta}$.

Proof. For the first identity, the log likelihood of the corpus equals the expected complete data log likelihood under the parse posterior plus the entropy of that posterior, which gives $\sum_w c_w \log P(w \mid \hat{\theta}) = \sum_i \hat{c}_i \log \hat{\theta}_i + H(r_{\hat{\theta}})$. At the fixed point $\hat{\theta}_i = \hat{c}_i / \hat{c}_A$, so the first term cancels the maximum likelihood term that $\mathcal{E}_{\text{approx}}$ subtracts. For the second identity, $\log Z_w(\bar{\theta})$ decomposes in the same way into $\sum_i \mathbb{E}_q[n_i] \log \bar{\theta}_i + H(q)$. The Kullback Leibler divergence between Dirichlet distributions is $\log B(\alpha) - \log B(\beta) + \sum_i (\beta_i - \alpha)(\psi(\beta_i) - \psi(B_A))$, where B is the multivariate beta function, and at the fixed point $\beta_i - \alpha = \mathbb{E}_q[n_i]$ and $\psi(\beta_i) - \psi(B_A) = \log \bar{\theta}_i$, so its last sum equals the first term of $\log Z$. What remains is $\log B(\beta) - \log B(\alpha) + H(q)$, and $\log B(\alpha + c) - \log B(\alpha)$ is $\log \text{DM}(c)$. $\square$

Both scores combine a Dirichlet integral with parse entropy, evaluated using the parse posterior and expected counts at different parameter settings. When the corpus is unambiguous under G , both entropies vanish and both sets of expected counts equal the fixed counts of the unique parses, so the two scores are identical. They are also identical whenever all parses of each string use the same multiset of productions, as with $S \rightarrow SS \mid a$, where the posterior over parses is uniform whatever the parameters. Outside these cases, equality is not guaranteed. Table 1 demonstrates a difference for the grammar $S \rightarrow SS \mid SSS \mid a \mid b \mid ab$ on the corpus $\{7: ab, 3: aba, 2: baba, 4: a, 1: ababa\}$. The difference there grows with α and changes sign, so the approximate evidence is neither a lower nor an upper bound on the variational bound. The experiments in Section 6 use approximate evidence throughout.

α	Iterations	$\mathcal{E}_{\text{approx}}$	$\mathcal{E}_{\text{vb}}$	Difference
0.1	8	-50.893	-50.907	+0.014
0.5	30	-47.437	-47.297	-0.141
2.0	30	-47.808	-46.391	-1.417

Table 1: The two scores on an ambiguous grammar, in nats. On the unambiguous running example and on $S \rightarrow SS \mid a$ the two scores agree to within 10^{-14} .

5 The running example and the effect of corpus size

The original paper uses the corpus $\{ab, aabb, aaabbb\}$ as its running example. Since the counts are not given there, we assign frequencies of 30, 15 and 5. On the search path shown in Figure 2, the initial grammar G_0 has a posterior of -72.35 nats and a description length of 30.8 bits, and chunking T_aT_b into X lowers the posterior to -74.23 , because the grammar gains a production and a nonterminal while still generating the same language. The merge of S with X then raises it to -62.51 , and once normalisation has removed the productions that the new recursive production makes redundant, the result is $S \rightarrow T_aST_b \mid T_aT_b$, which generates a^nb^n . Figure 2 of the original paper reaches the same grammar in four steps, chunking AB , then chunking AXB , and merging twice. The implementation removes $S \rightarrow T_aXT_b$ as soon as the merge makes it derivable, so one chunk and one merge are enough. Expectation maximisation gives the recursive production a probability of $1/3$, the maximum likelihood value for 25 recursive expansions among 75 uses of S in the corpus.

The original paper predicts that generalisation depends on the absolute corpus size as well as on relative frequencies. Multiplying every count by a constant multiplies the likelihood lost by any generalisation by the same constant, while the saving in the prior stays fixed. We evaluate this effect by scaling the total while retaining the ratio $6:3:1$. Table 2 reports the resulting scores. At every tested scale up to 200 samples, the search generalises to a^nb^n , with decreasing posterior gain as corpus size increases. At 300 samples and above, it accepts no edit and returns the memorising grammar. For this ratio and the default settings the threshold lies somewhere between 200 and 300 samples.

The frequency distribution also affects the outcome. Near uniform counts of 17, 17 and 16 total 50 samples but result in no accepted edit. The recursive grammar gives string lengths a geometric distribution, so its likelihood on the uniform corpus is -68.62 nats against -47.74 on the decaying one, and the likelihood loss exceeds the saving from the shorter description.

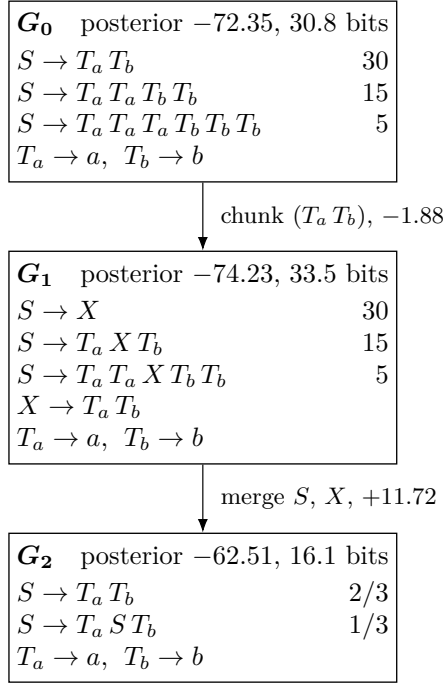


Figure 2: The accepted steps on the running example, with posteriors in nats and description lengths in bits. The counts in G_0 and G_1 are sample counts, and the values in G_2 are the fitted production probabilities. The chunk lowers the posterior by 1.88 nats and the merge raises it by 11.72.

Counts	Initial posterior	Final posterior	Recursive
6, 3, 1	-34.78	-23.51	yes
15, 7, 2	-47.28	-36.16	yes
30, 15, 5	-72.35	-62.51	yes
60, 30, 10	-117.94	-110.60	yes
120, 60, 20	-208.43	-206.42	yes
180, 90, 30	-298.63	-298.63	no
300, 150, 50	-478.74	-478.74	no
3000, 1500, 500	-4521.79	-4521.79	no

Table 2: Generalisation on the running example as the corpus grows with fixed proportions. The second row rounds the proportions down to integer counts.

6 Experiments

6.1 Setup

The evaluation covers eleven target grammars: the eight examples in Table 1 of the original paper, taken from Cook et al. [2], and three synthetic languages testing nesting and operator precedence. Following that table, the palindrome target is wcw^R with a centre marker, which is easier than the unmarked ww^R . For each target the driver samples 50 training strings and 200 held out strings from the target with every production equally likely, using seeds 1 and 1001, discards strings longer than seven symbols, and combines repeated strings into counts. Induction starts from the initial grammar of the training sample and uses the approximate evidence with $\alpha = 0.1$ and prior weight one, a beam of three, at most 25 rounds and a patience of five rounds, and once the search ends the production probabilities of the final grammar are refitted on the training sample. The original paper trained on the small sets of high probability strings used by Cook et al., with counts scaled to sum to 50, and chose best first search or a beam of width three

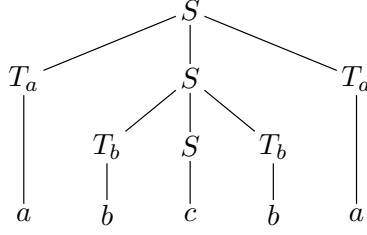


Figure 3: The Viterbi parse of *abcba* under the grammar induced for the palindromes.

or four separately for each language. The present setup matches the original corpus size, but its sampled corpora and search settings differ.

Evaluation compares generated languages, held out likelihood and recursion. The language comparison enumerates every string of length at most seven that each grammar generates and records whether the induced grammar generates every string of the target, what fraction of its own strings belong to the target, and whether the two sets are equal. The held out log likelihood is the average log probability per token of the held out strings, and it is $-\infty$ if any held out string has no parse. The final check records recursion, which permits an induced grammar to generate strings longer than those observed in training.

6.2 Results

Table 3 reports the outcome for each target. Every induced grammar parses all held out strings and matches the target language up to length seven, with precision one. It is recursive exactly when its target is recursive. The accepted search paths contain between one and seven steps, with an overall posterior improvement in every case. The largest improvements come on the arithmetic, basic English and nested parentheses targets, whose training samples contain the most distinct strings and whose initial grammars are consequently the longest.

Group	Target	$ \mathcal{N} $	$ P $	Initial	Final	Held out	Novel	Steps	Exact	Rec.
paper	parens	3	5	-124.81	-100.91	-0.438	2.5%	2	yes	yes
paper	a2n	2	3	-60.67	-54.43	-0.290	0%	2	yes	yes
paper	abn	3	4	-68.02	-58.54	-0.290	0%	2	yes	yes
paper	anbn	3	4	-76.14	-68.73	-0.336	0%	2	yes	yes
paper	palindrome	3	5	-198.88	-120.80	-0.717	7.5%	1	yes	yes
paper	addition	4	7	-203.57	-134.30	-1.120	20.5%	2	yes	yes
paper	shape	5	7	-251.23	-140.65	-0.646	12.0%	4	yes	yes
paper	basic_english [†]	11	22	-470.39	-286.80	-0.986	5.0%	7	yes	no
synthetic	nested_parens	3	5	-267.46	-149.87	-0.794	8.5%	1	yes	yes
synthetic	nested_anbn	3	4	-93.67	-68.16	-0.481	0%	1	yes	yes
synthetic	expr	4	8	-424.16	-223.34	-1.291	24.5%	3	yes	yes

Table 3: Results for each target. $|\mathcal{N}|$ and $|P|$ are the sizes of the induced grammar, Initial and Final are posteriors in nats, Held out is the average log probability per token, Novel is the share of held out mass made up of strings absent from the training sample, Steps counts accepted operators, and Exact and Rec. record language equality up to length seven and recursion. The row marked [†] is the result committed in the repository, which our rerun had not reproduced when this version was built.

Several induced grammars reproduce the target structure up to nonterminal renaming. The palindromes give $S \rightarrow c \mid T_a S T_a \mid T_b S T_b$, the running language gives $S \rightarrow T_a S T_b \mid T_a T_b$, and the nested parentheses give $S \rightarrow a \mid T(S T) \mid S S$. Figure 3 presents the most probable parse of *abcba* under the induced palindrome grammar, which has the same nested structure as the target.

For the arithmetic target, language recovery does not imply recovery of the intended structure. The target separates sums from products with the productions $S \rightarrow S + T \mid T$, $T \rightarrow T * F \mid F$

and $F \rightarrow (S) \mid a \mid b$, whereas the induced grammar is $S \rightarrow ST_*S \mid T_*(ST) \mid a \mid b$ with $T_* \rightarrow * \mid +$. The search merges the operator classes and collapses the three precedence levels into one nonterminal. The resulting grammar generates the same strings but is ambiguous and does not represent precedence. With the two languages equal, only the likelihood of the sample can tell the grammars apart, and here the difference in likelihood is too small to outweigh the much shorter description of the collapsed grammar.

The Novel column records a limitation of the held out evaluation. The languages a^{2n} , $(ab)^n$, $a^n b^n$ and the nested $a^n c b^n$ have only three or four strings of length at most seven, and the 50 training samples include all of them, so none of the 200 held out samples is a new string. Their likelihood measures fit to strings already represented in training. For the same reason, comparing languages up to length seven cannot tell the induced grammar apart from a grammar that memorises the training set on these four targets. For these targets, evidence of generalisation comes from the induced recursive grammar, which generates the target strings beyond the observed lengths. Only the addition and arithmetic targets put a substantial share of held out mass on novel strings, about a fifth and a quarter respectively.

References

- [1] J. K. Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1), page S132, 1979. <https://doi.org/10.1121/1.2017061>
- [2] C. M. Cook, A. Rosenfeld and A. R. Aronson. Grammatical inference by hill climbing. *Information Sciences*, 10(1), pages 59 to 80, 1976. [https://doi.org/10.1016/0020-0255\(76\)90061-X](https://doi.org/10.1016/0020-0255(76)90061-X)
- [3] A. P. Dempster, N. M. Laird and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1), pages 1 to 22, 1977. <https://doi.org/10.1111/j.2517-6161.1977.tb01600.x>
- [4] K. Kurihara and T. Sato. Variational Bayesian grammar induction for natural language. In *Grammatical Inference, Algorithms and Applications (ICGI 2006)*, Lecture Notes in Computer Science, pages 84 to 96. Springer, 2006. https://doi.org/10.1007/11872436_8
- [5] C. Lanczos. A precision approximation of the gamma function. *Journal of the Society for Industrial and Applied Mathematics, Series B, Numerical Analysis*, 1, pages 86 to 96, 1964. <https://doi.org/10.1137/0701008>
- [6] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4(1), pages 35 to 56, 1990. [https://doi.org/10.1016/0885-2308\(90\)90022-X](https://doi.org/10.1016/0885-2308(90)90022-X)
- [7] B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60, pages 94 to 112, 2014. <https://doi.org/10.1016/j.jsc.2013.09.003>
- [8] A. Stolcke and S. Omohundro. Inducing probabilistic grammars by Bayesian model merging. In *Grammatical Inference and Applications (ICGI 1994)*, Lecture Notes in Computer Science, pages 106 to 118. Springer, 1994. https://doi.org/10.1007/3-540-58473-0_141, also <https://arxiv.org/abs/cmp-lg/9409010>