

# Appendix

## A Hardware and Software

Our experiments were conducted on Ubuntu 22.04 LTS nodes with Intel Xeon Gold 6230 CPUs (2.10 GHz, 10 cores, 20 threads allocated) and 384 GB RAM. For GPU-accelerated workloads, we provisioned 8x NVIDIA GeForce RTX 2080 Ti GPUs. Our implementation is based on Python 3.10.12, PyTorch 2.6.0+cu124, AFL++ 4.00c and LLVM 14.0.0.

## B Hyperparameters

For language-model decoding, we set temperature to 1.0, top-p to 0.9, and top-k to 0 to allow sampling from the full token vocabulary. We limited the maximum number of newly generated tokens to 512 for XML, and 1024 for SQL .test scripts.

## C Model Checkpoint

We use the Llama-3.1-8B model checkpoint provided by Hugging Face (commit 0e9e39f): <https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct>.

## D Fuzzing Experiments Details

### D.1 Benchmarks

Table 1 summarizes the libraries, versions, and seed formats for each target.

Table 1: Benchmarks, versions, and seed formats.

Target	Library	Version	Seed format
XML <sup>[17, 16]</sup>	libxml2	2.15.0	.xml
SQL <sup>[51]</sup>	sqlite	3.49.2	.test

### D.2 Parse-Tree Illustration

Figure 4 shows a comprehensive parse tree for a SQLite test case, derived from the grammar in Figure 1b.

### D.3 Prompts and Constraints

For all benchmarks, we use a standard in-context learning format where the prompt consists of a single (specification, solution) pair, followed by a new specification for which the model must generate a solution. A representative prompt for the XML benchmark is shown in Figure 5a, along with its corresponding grammar in Figure 5b.

### D.4 Fuzzing Protocol and Environment

All fuzzing experiments were conducted using AFL++ 4.00c on the hardware and software setup described in Appendix A. Each (benchmark, method) pair was evaluated in  $N = 5$  independent, single-instance AFL++ runs of exactly 3600 s (one hour). We set ‘AFL\_RANDOM\_SEED’ to  $42 + i$ , ( $i = 1 \dots 5$ ) for reproducibility and configure standard environment variables to ensure non-interactive execution. All other AFL++ parameters remained at defaults to isolate the impact of seed corpus quality. Complete build and execution scripts are provided in the supplementary materials.

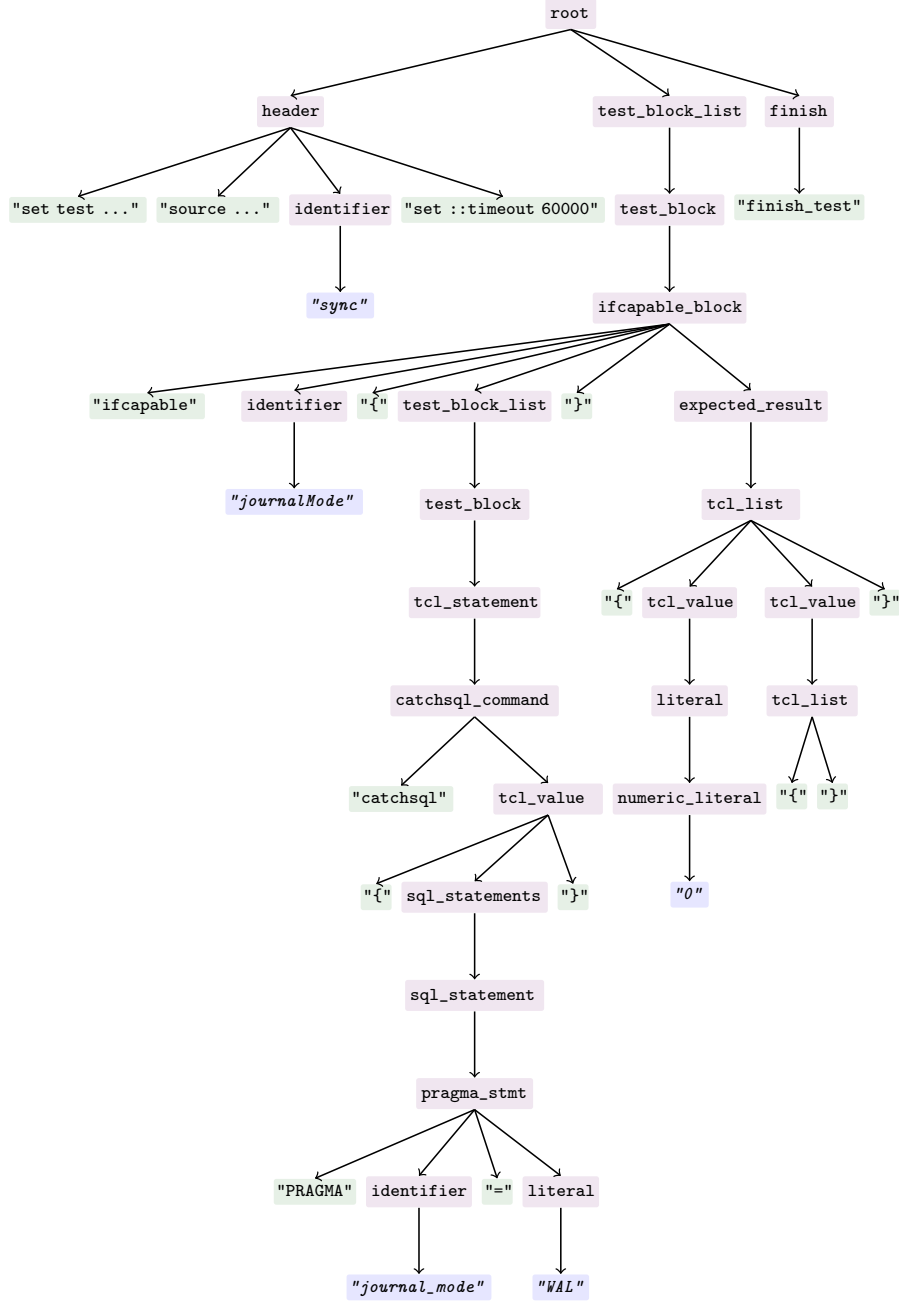


Figure 4: Condensed parse tree for the example SQLite .test script used in Figure 1. Purple boxes denote non-terminals, green boxes denote grammar terminals, and blue italics show literal terminal values substituted during this derivation. Subtrees unrelated to the header and the first do\_execsql\_test block are elided for brevity.

## 543 D.5 Coverage Measurement via LLVM Instrumentation

544 We measured branch coverage using LLVM’s instrumentation toolchain  
 545 (-fprofile-instr-generate -fcoverage-mapping), which adds  $\leq 2\%$  runtime over-  
 546 head. Raw profiles were collected during execution and aggregated post-trial using llvm-profdata and  
 547 llvm-cov.

Question 1:  
Generate a single, short but complex XML document.  
Include a variety of XML features and various  
markup declarations. Do not reuse previous solutions.

Solution 1:  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE document [  
 <!ENTITY note "Take care!">  
<document xmlns="http://example.com/schema"... >  
 ...  
</document>

Question 2:  
Generate a single, short but complex XML document.  
Include a variety of XML features and various  
markup declarations. Do not reuse previous solutions.

Solution 2:

```

root ::=
  (prolog)? ws?
  (doctype ws)?
  element

prolog ::=
  xmldecl (ws misc)*

...

element ::=
  starttag content endtag
  | emptyelement

endtag ::=
  "</" name ">"

...

content ::=
  (chardata
   | element
   | comment
   | pi
   | cdata)*

comment ::=
  "<!--" [^~]* "-->" ws

...

ws ::= [white_space]*

```

(a) Prompt

(b) Grammar

Figure 5: (a) Prompt given to a LM to generate seed test cases for fuzzing the XML parser. (b) Simplified version of the XML grammar written in EBNF notation. The goal of the problem is to generate multiple diverse seeds that trigger different code paths in the library being tested.

548 **Rationale.** We report branch coverage rather than crash counts because the experiment isolates  
549 *seed quality* — all methods receive the same fixed prompt per benchmark, so coverage is a good  
550 measure on how their seeds exercise the code.

## 551 D.6 Rejection Sampling Acceptance Rates

552 We quantified the viability of rejection sampling under the same grammar constraints used by our  
553 MCMC framework in Section 4.2. Across 500 attempted samples per benchmark, the proportion of  
554 syntactically and semantically valid outputs was consistently below 1%.

## 555 D.7 Branch Coverage and KL Divergence for XML

556 Figure 6 shows the same analysis illustrated in the main text for SQLite, but for the XML benchmark.

## 557 D.8 Ablation Study

558 To isolate the effect of (i) the proposal family (Priority, Restart, Uniform) and (ii) the number of steps  
559  $k \in \{2, 5, 10\}$ , we plot each family separately against the heuristic baseline (**GCD**).<sup>1</sup> Figures 7–8  
560 reveal two consistent patterns.

- 561 1. **Number of steps matters, but saturates.** Coverage grows monotonically with  $k$ ; however,  
562  $k=5$  already captures  $\geq 95\%$  of the gain realised by  $k=10$  on both benchmarks.
- 563 2. **All MCMC variants beat GCD.** Even the weakest setting ( $k=2$ ) surpasses GCD’s final  
564 coverage by 4-5%, demonstrating that MCMC proposals yield coverage gains over heuristic  
565 constrained decoding — even with very few sampling steps.

<sup>1</sup>Grammarinator is omitted for clarity; its curve lies far below all others and does not alter the ordering.

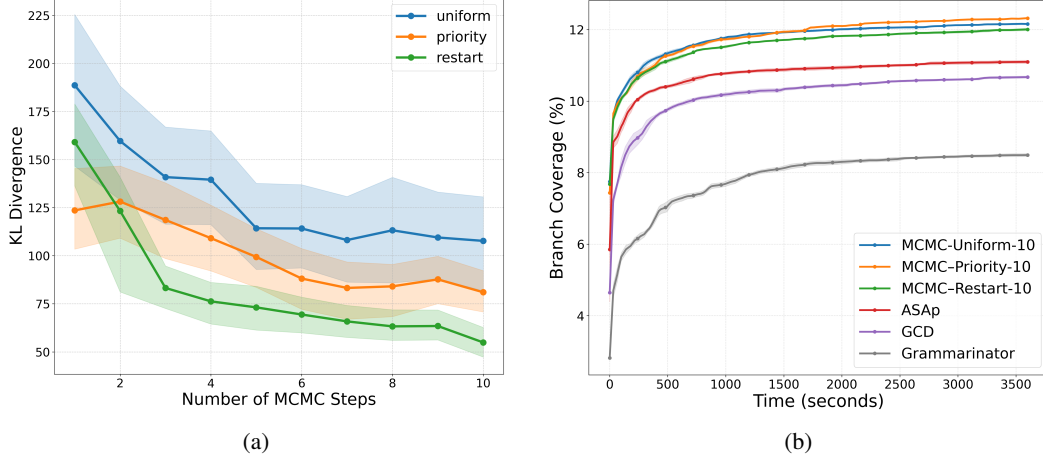


Figure 6: (a) KL divergence (mean  $\pm$  95 % CI, hundred runs per approach) for MCMC with varying number of steps for the XML Benchmark. (b) Branch coverage over time (mean  $\pm$  95 % CI, five trials per method) for the XML Benchmark.

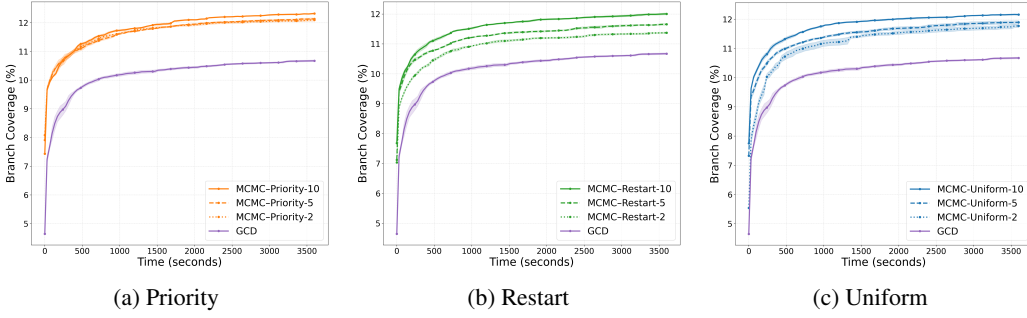


Figure 7: **XML**: branch-coverage ablation. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

## 566 D.9 Function and Line Coverage

567 To corroborate the branch-coverage trends reported in the main text, we additionally measure both

- 568 • **Line coverage**: the fraction of source lines executed (Figures 11 and 12), and
- 569 • **Function coverage**: the fraction of instrumented functions executed (Figures 9 and 10).

570 using the same `llvm-cov` instrumentation described in Appendix D.5.

## 571 D.10 Overall Coverage Results

572 Tables 2 and 3 show that the improvements achieved by our grammar-based MCMC sampler on  
 573 branch coverage translate consistently to both function and line coverage across both the SQL and  
 574 XML benchmarks.

## 575 E Properties and Proofs

576 In this section, we formalize and prove the two key properties of our sampler (Alg. 1), constraint  
 577 satisfying (Thm. 1) and monotonically converging (Thm. 3).

578 The first property follows directly from the procedure in Alg. 1.

579 **Theorem 1** (Constraint Satisfying). *For any LM  $P$ , any grammar  $G$ , any chain length  $k$ , and any*  
 580 *truncation distribution  $p_{\text{POS}}$ , the result of Alg. 1 is always inside  $\mathcal{L}(G)$ .*

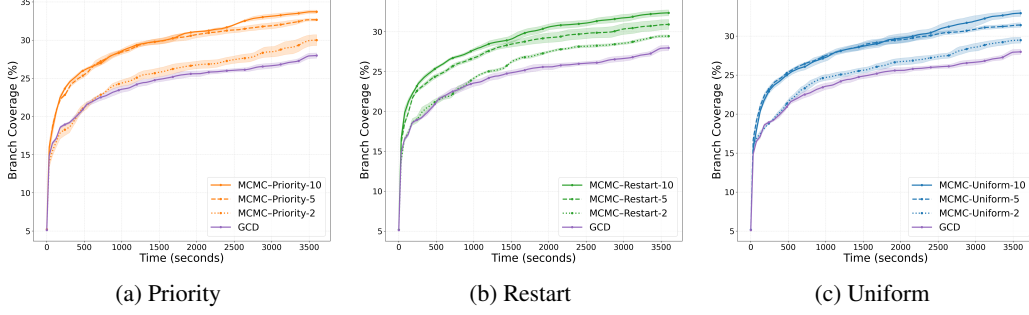


Figure 8: **SQL**: branch-coverage ablation. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

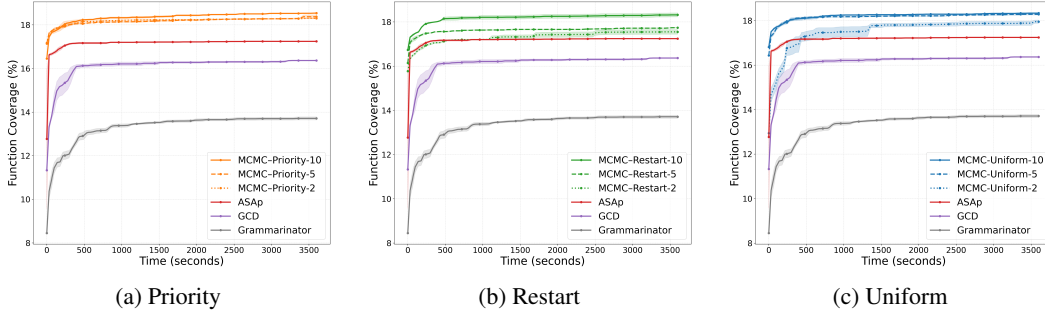


Figure 9: **XML**: function-coverage. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

581 *Proof.* The output of Alg. 1 can only be generated from either Line 1 or Line 11, both of which call  
 582 the GCD procedure. Since GCD samples only from the constrained language  $\mathcal{L}(G)$ , the result of  
 583 Alg. 1 must also fall within  $\mathcal{L}(G)$ .  $\square$

584 As for monotonically converging, we prove it by applying the following theorem for Markov chains.

585 **Theorem 2** (Thm. 5.6.6 in [12]). *Let  $p$  be a Markov chain with countable states, and let  $\|q, q'\|_{\text{TV}}$*   
 586 *denotes the total variance distance of two distributions, i.e.,  $\frac{1}{2} \sum_x |q(x) - q'(x)|$ .*

587 *When  $p$  is irreducible, aperiodic, and has stationary distribution  $\pi$ , then for any state state  $x$*   
 588  *$\|p^k(\cdot | x), \pi\|_{\text{TV}}$  will converge to 0 as  $k$  approaches to  $\infty$ .*

589 **Theorem 3** (Monotonically Converging). *For any LM  $P$ , any grammar  $G$ , and any truncation*  
 590 *distribution  $p_{\text{POS}}$ , if  $p_{\text{POS}}^w(0) > 0$  for all sequences  $w \in \mathcal{L}(G)$ , then the output distribution of Alg. 1*  
 591 *will monotonically converge to  $P^G$  as the chain length  $k$  approaches to infinite, as shown below.*

$$\lim_{k \rightarrow \infty} \|P_k^{\mathcal{O}}, P^G\|_{\text{TV}} = 0 \quad (3)$$

$$\forall k, \|P_k^{\mathcal{O}}, P^G\|_{\text{TV}} \geq \|P_{k+1}^{\mathcal{O}}, P^G\|_{\text{TV}} \quad (4)$$

592 where  $P_k^{\mathcal{O}}$  denotes the output distribution of Alg. 1 when the chain length is  $k$ .

593 *Proof.* We prove the **convergence** (Eq. 3) by applying Thm. 2 to our case, by verifying that the  
 594 Markov chain constructed in Alg. 1 satisfies all prerequisites of Thm. 2.

595 1. (Countable states) In our Markov chain, the state set comprises all sequences with non-zero  
 596 probability in  $P^G$ , denoted as  $S$ . This set is countable since the set of all sequences is  
 597 countable.

598 2. (Irreducibility) Let  $q$  be the proposal distribution of our Markov chain. We start by showing  
 599 that  $q(y | x) > 0$  for all  $x, y \in S$ . Consider the event where the empty prefix is selected  
 600 in Line 10, and  $y$  is selected by GCD in Line 11. The probability of this event is  $p_{\text{POS}}^x(0) \cdot$

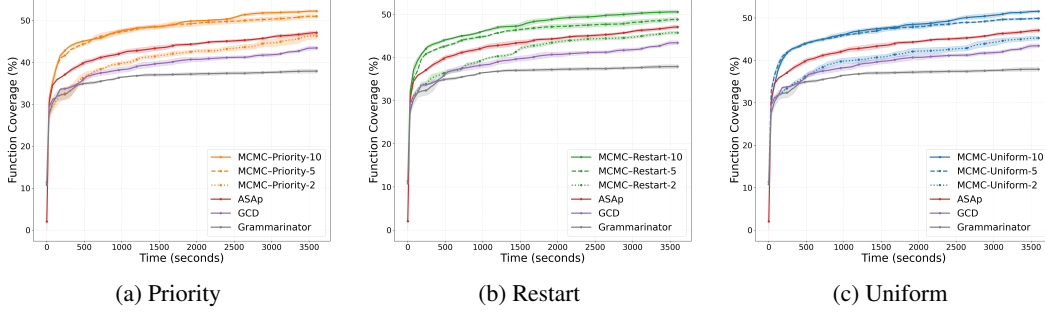


Figure 10: **SQL**: function-coverage. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

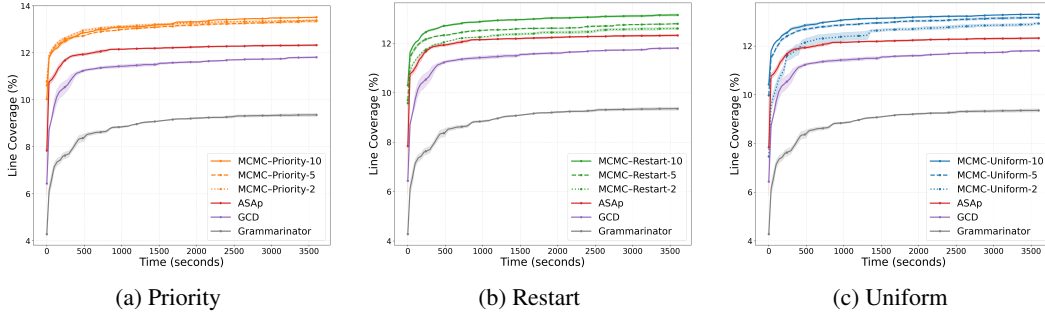


Figure 11: **XML**: line-coverage. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

601  $P_{\text{GCD}}(y)$ , which is non-zero. On the other hand,  $q(y | x)$  is no smaller than this probability,  
 602 hence it must also be non-zero.

603 Then, by the definition of the transition probability  $p$  in the Metropolis-Hastings algorithm<sup>2</sup>

$$\forall x, y \in S, \quad p(y | x) \geq q(y | x) \cdot \alpha(x, y) = q(y | x) \cdot \max \left\{ 1, \frac{P(y)q(x | y)}{P(x)q(y | x)} \right\}.$$

604 All values on the right-hand side are positive, so  $p(y | x)$  must also be positive, implying  
 605 the irreducibility of the Markov chain.

606 3. (Aperiodicity) By the above analysis,  $p(x | x) > 0$  for any state  $x$ , implying aperiodicity.

607 4. (Stationary distribution) The Metropolis-Hastings algorithm ensures that the target distribu-  
 608 tion  $P^{\mathcal{G}}$  is a stationary distribution of the constructed Markov chain.

609 Hence, all prerequisites of Thm. 2 are satisfied, then Eq. 3 follows directly from it.

---

<sup>2</sup>The greater-than part of the first inequality captures the special case where  $x = y$ :

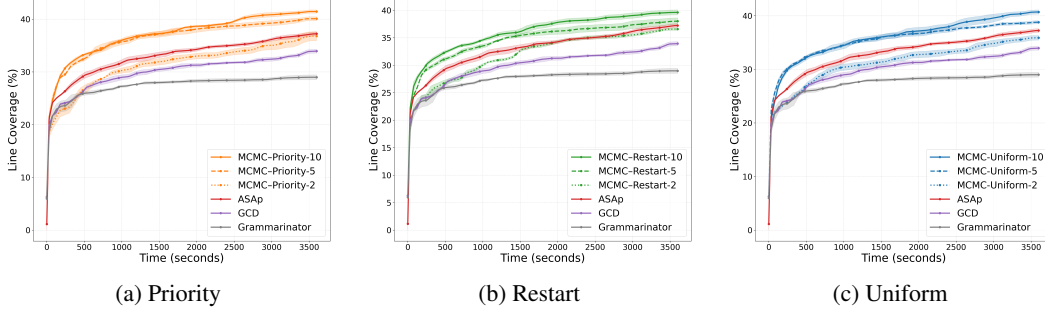


Figure 12: **SQL**: line-coverage. Line style encodes the number of steps  $k \in \{2, 5, 10\}$  (dotted, dashed, solid).

Table 2: SQL benchmark coverage (mean  $\pm$  95 % CI) over five trials, and relative gain in branch coverage versus GCD, for all  $k$ . Best entries in green.

Method	$k$	Branch (%)	Function (%)	Line (%)	$\Delta$ vs GCD (%)
<b>ASAP</b>	—	29.91(50)	47.07(83)	37.23(72)	5.8
<b>GCD</b>	—	28.26(68)	43.87(96)	34.28(73)	0.0
<b>Grammarinator</b>	—	25.04(91)	39.89(96)	30.52(94)	−11.4
<b>MCMC-Prefix</b>					
	2	29.47(75)	45.25(96)	35.88(84)	4.3
	5	31.41(57)	49.88(48)	38.78(59)	11.2
	10	32.93(83)	51.53(71)	40.69(77)	16.5
<b>MCMC-Priority</b>					
	2	29.99(140)	46.34(164)	36.78(170)	6.1
	5	32.66(32)	50.99(82)	40.09(74)	15.6
	10	33.70(44)	52.22(58)	41.45(62)	19.3
<b>MCMC-Restart</b>					
	2	29.45(32)	45.75(70)	36.59(45)	4.2
	5	30.91(123)	48.83(89)	38.02(120)	9.4
	10	32.36(74)	50.57(92)	39.62(90)	14.5

610 Then, we prove the **monotocity** by the following derivation, where  $p(w \mid w')$  denotes the probability  
611 for our Markov chain to move from  $w'$  to  $w$ .

$$\begin{aligned}
\|P_{k+1}^{\mathcal{O}}, P^{\mathcal{G}}\|_{\text{TV}} &= \frac{1}{2} \sum_w |P_{k+1}^{\mathcal{O}}(w) - P^{\mathcal{G}}(w)| \\
&= \frac{1}{2} \sum_w \left| \sum_{w'} (P_k^{\mathcal{O}}(w') - P^{\mathcal{G}}(w')) \cdot p(w \mid w') \right| \\
&\leq \frac{1}{2} \sum_w \sum_{w'} |P_k^{\mathcal{O}}(w') - P^{\mathcal{G}}(w')| \cdot p(w \mid w') \\
&= \frac{1}{2} \sum_{w'} |P_k^{\mathcal{O}}(w') - P^{\mathcal{G}}(w')| \left( \sum_w p(w \mid w') \right) \\
&= \frac{1}{2} \sum_{w'} |P_k^{\mathcal{O}}(w') - P^{\mathcal{G}}(w')| = \|P_k^{\mathcal{O}}, P^{\mathcal{G}}\|_{\text{TV}}
\end{aligned}$$

612

□

## 613 F Benchmarks by Park et al. [34]

614 We evaluate the convergence properties of Grammar-Aligned MCMC Sampling empirically on  
615 the benchmark tasks proposed by Park et al. [34]. Fig. 13 relates the  $KL(P^{\mathcal{G}} \| GCD)$  and

Table 3: XML coverage (mean  $\pm$  95 % CI) over five trials, and relative gain in branch coverage versus GCD, for all  $k \in \{2, 5, 10\}$ . Best branch performer in green.

Method	$k$	Branch (%)	Function (%)	Line (%)	$\Delta$ vs GCD (%)
<b>ASAP</b>	—	11.10(9)	17.24(3)	12.33(7)	4.0
<b>GCD</b>	—	10.67(5)	16.36(2)	11.81(6)	0.0
<b>Grammarinator</b>	—	8.49(8)	13.71(14)	9.36(13)	-20.4
<b>MCMC-Prefix</b>					
	2	11.77(18)	17.95(14)	12.93(12)	10.3
	5	11.89(12)	18.28(6)	13.17(9)	11.4
	10	12.16(5)	18.33(3)	13.30(4)	14.0
<b>MCMC-Priority</b>					
	2	12.10(20)	18.30(13)	13.38(10)	13.4
	5	12.62(19)	19.11(17)	13.88(12)	18.2
	10	12.81(5)	19.28(5)	14.05(6)	20.0
<b>MCMC-Restart</b>					
	2	11.37(16)	17.55(12)	12.61(11)	6.6
	5	11.66(14)	17.74(7)	12.80(10)	9.4
	10	12.00(6)	18.32(9)	13.15(9)	12.5

616  $KL(P^G \| \text{MCMC-T}(k))$  for  $k = 10$ , for  $T \in \{\text{Uniform, Priority, Restart}\}$ . Each point represents a  
617 single task. Points below the diagonal indicate tasks where MCMC approximates  $P^G$  better than  
618 GCD. Fig. 14 displays the same information, but for ASAP(10) instead of GCD.

619 Fig. 15, Fig. 16, Fig. 17 compare the distance to  $P^G$  for ASAP( $k$ ) and MCMC-T( $k$ ), as  $k$  increases,  
620 across all the benchmark tasks. Lower KL-Divergence indicates a better approximation to  $P^G$ .

621 GCD, MCMC( $k$ ) and ASAP( $k$ ) are all approximated using 100 samples, and  $P^G$  is approximated  
622 using all the samples acquired during the runs of MCMC and ASAP. A 95% confidence band is  
623 shown for convergence plots, computed via Bootstrapping.

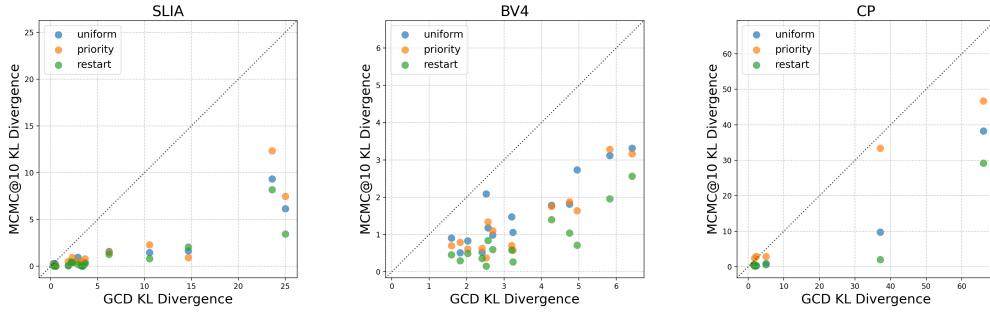


Figure 13: KL-Divergence for GCD vs MCMC( $k = 10$ ) by subset

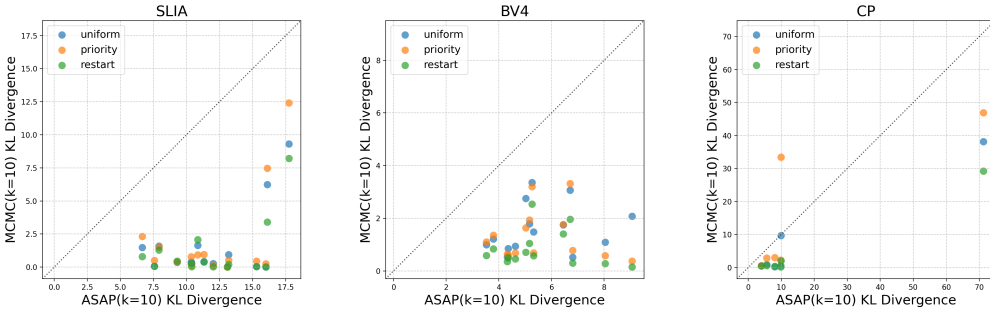


Figure 14: KL-Divergence for ASAP( $k = 10$ ) vs MCMC( $k = 10$ ) by subset

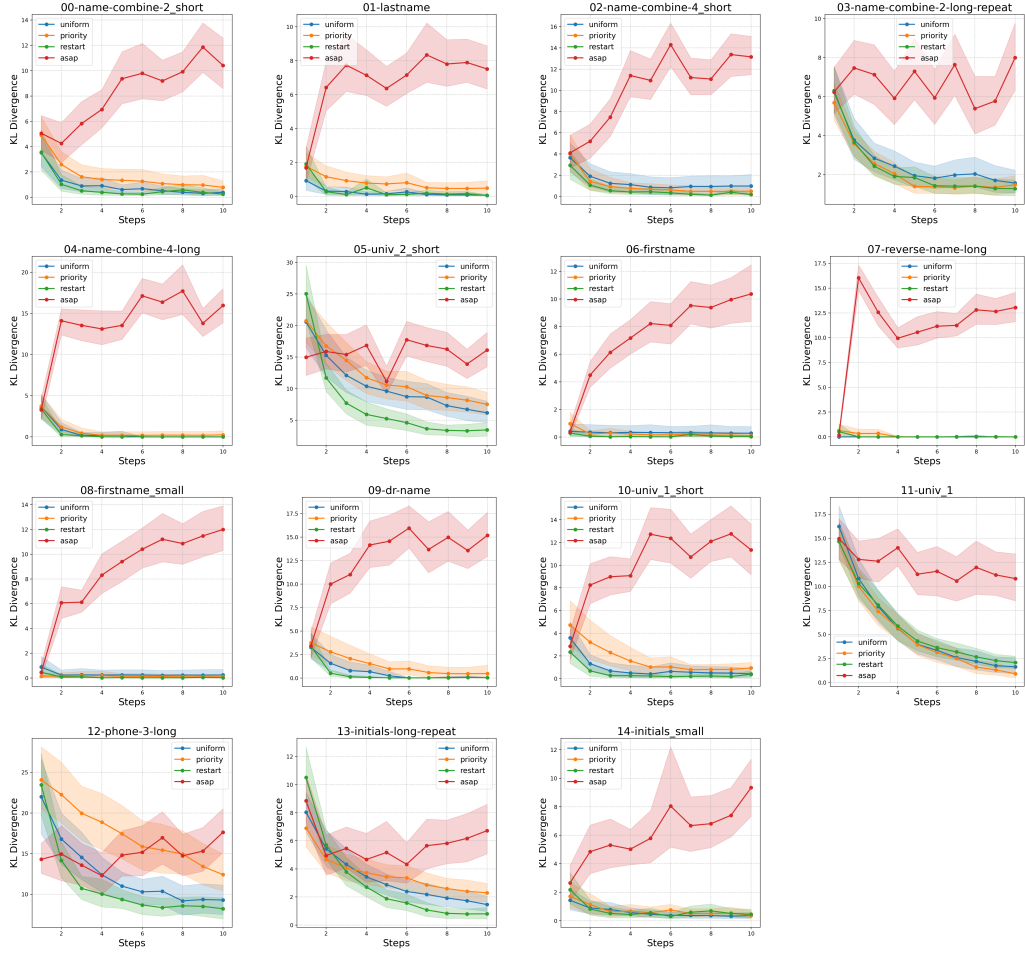


Figure 15: KL-Divergence for  $\text{ASAp}(k)$  and  $\text{MCMC}(k)$  in SLIA subset

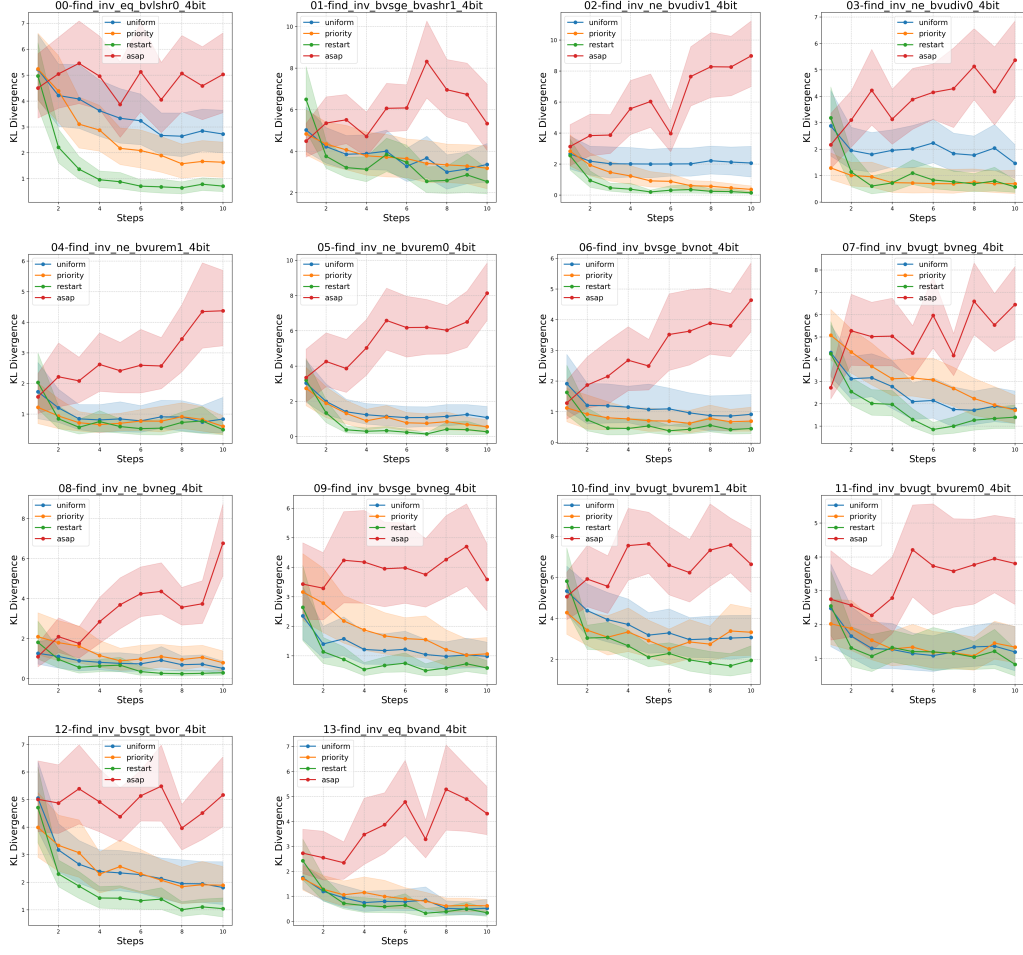


Figure 16: KL-Divergence for  $ASAp(k)$  and  $MCMC(k)$  in BV4 subset

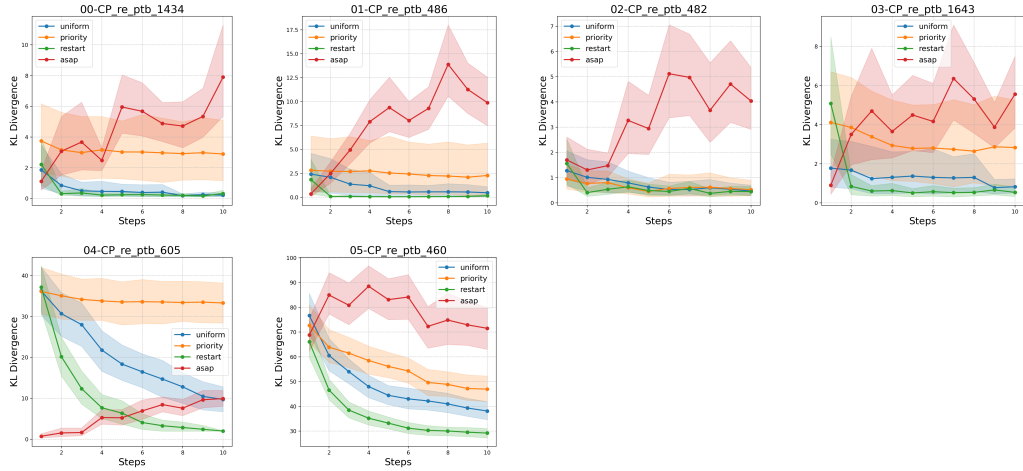


Figure 17: KL-Divergence for  $ASAp(k)$  and  $MCMC(k)$  in CP subset