

A Word Sampler for Well-Typed Functions

Breandan Mark Considine

January 11, 2026

Formal languages & type theory

$$\underbrace{\sigma \in \mathcal{L}(G) \Leftrightarrow \exists V. V \Rightarrow_G^* \sigma}_{\text{membership / parse tree}} \Leftrightarrow \underbrace{\exists \tau. (\Gamma \vdash e : \tau)}_{\text{type checking / proof tree}}$$

$$\underbrace{(W \rightarrow XZ) \in P}_{\text{grammar production}} \Leftrightarrow \underbrace{\frac{\Gamma \vdash x : X \quad \Gamma \vdash z : Z}{\Gamma \vdash xz : W}}_{\text{typing judgment}}$$

$$\underbrace{\mathcal{L}(G) \neq \emptyset \Leftrightarrow \exists \sigma. S \Rightarrow_G^* \sigma}_{\text{non-emptiness / generation}} \Leftrightarrow \underbrace{\exists e. (\Gamma \vdash e : \tau)}_{\text{type inhabitation / synthesis}}$$

Goal: Given a set of typing judgements and a typing context (Γ), design a grammar, G , s.t. $\forall \sigma \in \Sigma^{<n} \exists \tau . \sigma \in \mathcal{L}(G) \iff \Gamma \vdash \sigma : \tau$.

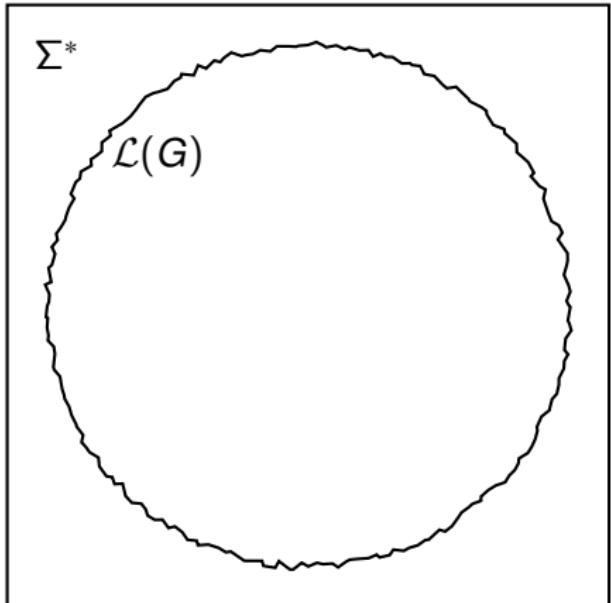
Programming language [in]approximability

- ▶ Σ^* : all words over Σ

$$\Sigma^*$$

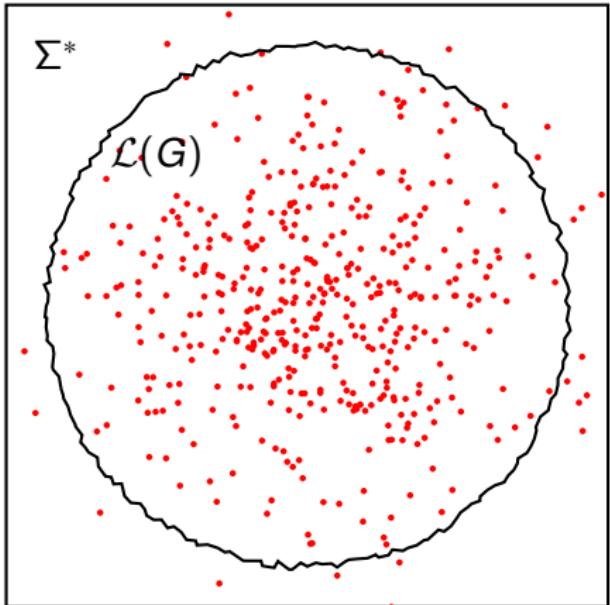
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid



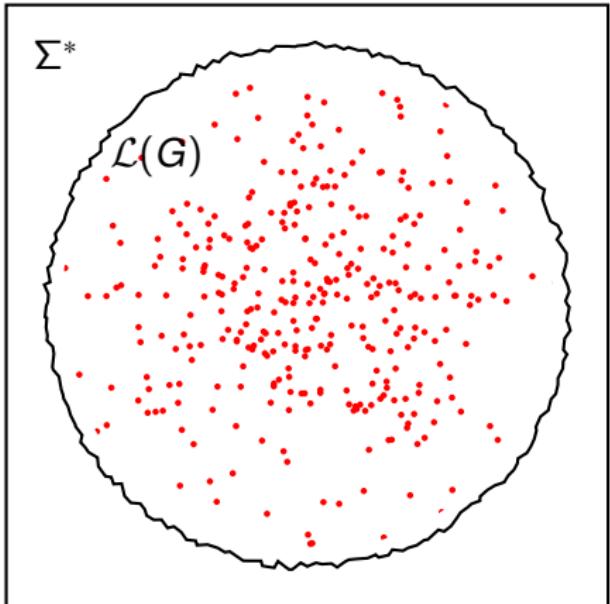
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \leftarrow \Sigma^*$



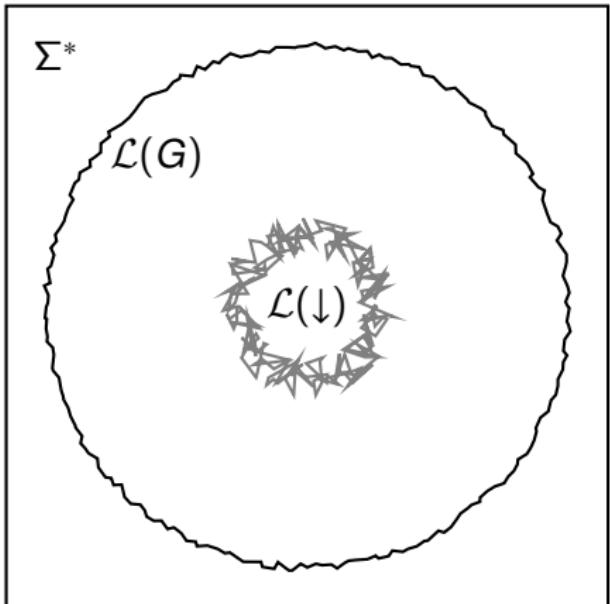
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \rightsquigarrow \Sigma^*$
- ▶ Guidance: $\sigma \rightsquigarrow \mathcal{L}(G)$



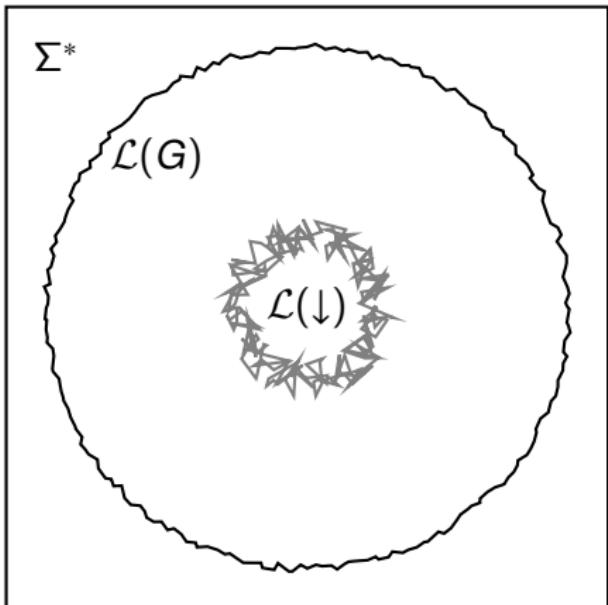
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \rightsquigarrow \Sigma^*$
- ▶ Guidance: $\sigma \rightsquigarrow \mathcal{L}(G)$
- ▶ $\mathcal{L}(\downarrow)$: halting programs



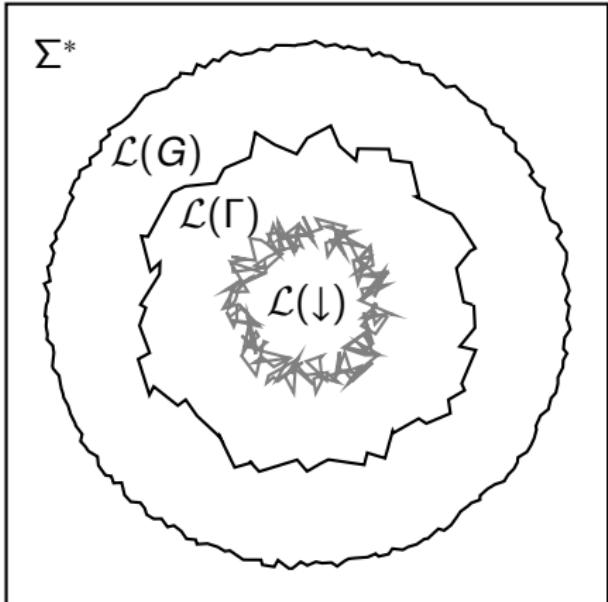
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \rightsquigarrow \Sigma^*$
- ▶ Guidance: $\sigma \rightsquigarrow \mathcal{L}(G)$
- ▶ $\mathcal{L}(\downarrow)$: halting programs
- ▶ Tighter approximations require ever-increasing expressive power



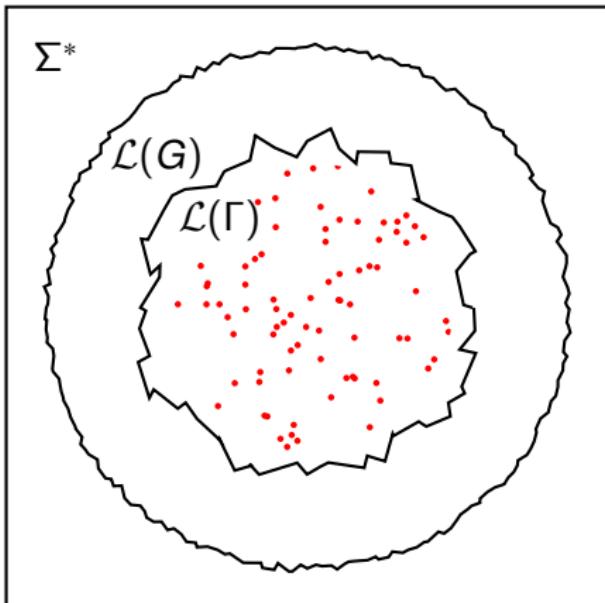
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \rightsquigarrow \Sigma^*$
- ▶ Guidance: $\sigma \rightsquigarrow \mathcal{L}(G)$
- ▶ $\mathcal{L}(\downarrow)$: halting programs
- ▶ Tighter approximations require ever-increasing expressive power
- ▶ $\mathcal{L}(\Gamma)$: type-safe programs



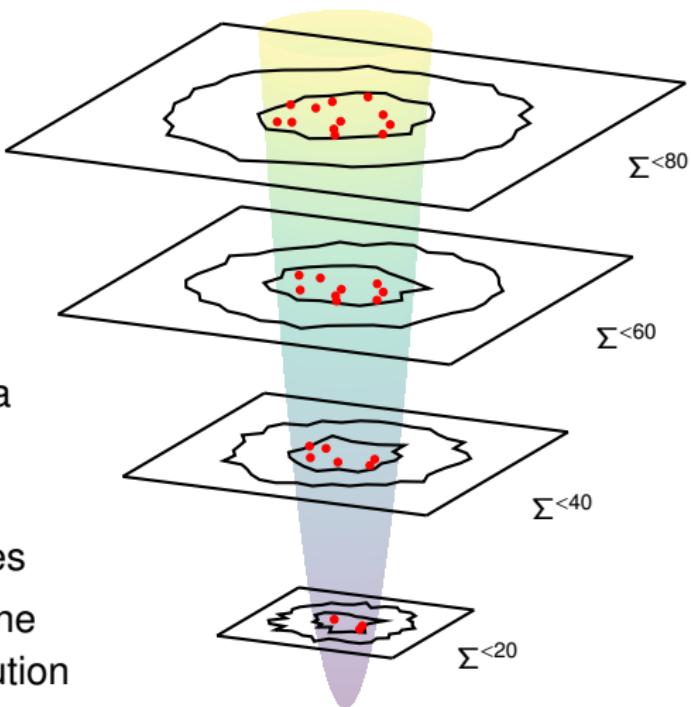
Programming language [in]approximability

- ▶ Σ^* : all words over Σ
- ▶ $\mathcal{L}(G)$: syntactically valid
- ▶ Most LLMs: $\sigma \rightsquigarrow \Sigma^*$
- ▶ Guidance: $\sigma \rightsquigarrow \mathcal{L}(G)$
- ▶ $\mathcal{L}(\downarrow)$: halting programs
- ▶ Tighter approximations require ever-increasing expressive power
- ▶ $\mathcal{L}(\Gamma)$: type-safe programs
- ▶ Typesafe: $\sigma \rightsquigarrow \mathcal{L}(\Gamma)$



Stratified sampling with finite model theory

- ▶ But $\mathcal{L}(\Gamma)$ is infinite
- ▶ Consider finite models
- ▶ Isolate key complexity parameters of interest
- ▶ Embed description into a context-free grammar
- ▶ Disintegrate into fixed-parameter tractable slices
- ▶ Sample uniformly from the exact conditional distribution



High-level grammar embedding recipe

- ▶ Fix a finite type universe \mathbb{T} and an ambient global context Γ
- ▶ Decorate vanilla nonterminals with a typing annotation, $E[\tau]$
- ▶ Each typing judgment becomes a schema for constructing a family of synthetic productions, each instantiated with $\tau : \mathbb{T}$

Syntax:
$$\frac{\Gamma \vdash e_1 : \tau_1, \dots, e_m : \tau_m \quad \Phi(\Sigma, \tau_1, \dots, \tau_m) : \tau}{(E[\tau] \rightarrow \Phi(\Sigma, \tau_1, \dots, \tau_m)) \in P_\Gamma}$$

Names: $\Gamma \vdash e : \tau \Rightarrow (E[\tau] \rightarrow e) \in P_\Gamma$

Functions:
$$\frac{\Gamma \vdash f : (\tau_1, \dots, \tau_k) \rightarrow \tau}{(E[\tau_1] \rightarrow f \ (E[\tau_1] \ , \ \dots \ , \ E[\tau_k])) \in P_\Gamma}$$

Example language: simply typed function syntax

```
FUN ::= fun f0 ( PRM ) : T = EXP
PRM ::= PID : T | PRM , PID : T
EXP ::= `N | `B | PID | INV | IFE | OPX
OPX ::= ( EXP OPR EXP )
IFE ::= if EXP { EXP } else { EXP }
INV ::= FID ( ARG )
ARG ::= EXP | ARG , EXP
OPR ::= + | * | < | ==
PID ::= p1 | ... | pk
FID ::= f0 | f1 | ... | fn
`B ::= true | false
`N ::= 1 | 2 | 3 | ...
```

Type universe: Finite T with two primitive types (e.g., B, N, \dots)

Ambient context: Γ maps $f_{} : (\tau_1, \dots, \tau_m) \rightarrow \tau$.

Expression fragment: static semantics

$$\frac{\Gamma \vdash e_c : \mathbb{B} \quad \Gamma \vdash e_T : \tau \quad \Gamma \vdash e_\perp : \tau}{\Gamma \vdash \text{if } e_c \{ e_T \} \text{ else } \{ e_\perp \} : \tau} \text{ IFE}$$

$$\frac{\Gamma \vdash f_? : (\tau_1, \dots, \tau_m) \rightarrow \tau \quad \Gamma \vdash e_i : \tau_i \quad \forall i \in [1, m]}{\Gamma \vdash f_? (e_1, \dots, e_m) : \tau} \text{ INV}$$

$$\frac{\delta_{OPR}(\odot, \tau, \tau') = \hat{\tau} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau'}{\Gamma \vdash (e_1 \odot e_2) : \hat{\tau}} \text{ OPX}$$

Where the operator typing function $\delta_{OPR} : \Sigma_{OPR} \times \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T}$ returns:

$$\delta_{OPR}(\odot, \tau, \tau') = \begin{cases} \mathbb{B} & \odot = <, \tau = \tau' = \mathbb{B} \\ \mathbb{N} & \odot \in \{+, *\}, \tau = \tau' = \mathbb{N} \\ \mathbb{B} & \odot = ==, \tau = \tau' \end{cases}$$

Embedding the type checker (I)

Grammar: $\langle \Sigma, V, P \subset V \times (V \cup \Sigma)^*, S \in V \rangle \Rightarrow \langle \Sigma_\Gamma, V_\Gamma, P_\Gamma, V_\Gamma, S_\Gamma \rangle$

Decorated nonterminals: $\text{EXP}[\tau, \pi] \quad (\tau \in \mathbb{T}, \pi \equiv (\vec{\tau} \rightarrow \dot{\tau}))$

Provide: k , the maximum arity, and \mathbb{T} , the type universe.

$$\frac{\langle \vec{\tau}, \dot{\tau} \rangle \in \mathbb{T}^{0..k} \times \mathbb{T} \quad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(S_\Gamma \rightarrow \text{fun } f\emptyset \left(\prod_{i=1}^{|\vec{\tau}|} (p_i : \vec{\tau}_i) \right) : \dot{\tau} = \text{EXP}[\dot{\tau}, \vec{\tau} \rightarrow \dot{\tau}] \right) \in P_\Gamma} \text{FUN}_\varphi$$

$$\frac{\text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \in V_\Gamma \quad \tau = \dot{\tau} \quad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(\text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \rightarrow f\emptyset \left(\prod_{i=1}^{|\vec{\tau}|} \text{EXP}[\vec{\tau}_i, \vec{\tau} \rightarrow \dot{\tau}] \right) \right) \in P_\Gamma} \text{REC}_\varphi$$

$$\frac{\text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \in V_\Gamma \quad \exists i. \vec{\tau}_i = \tau}{\left(\text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \rightarrow \text{pi} \right) \in P_\Gamma} \text{PID}_\varphi \quad \frac{\text{EXP}[\tau, \pi] \in V_\Gamma \quad _ : \mathbb{B} \mid \mathbb{N}}{\left(\text{EXP}[\tau, \pi] \rightarrow _ \right) \in P_\Gamma} \text{rT}_\varphi$$

Embedding the type checker (II)

$$\frac{\text{EXP}[\tau, \pi] \in V_\Gamma \quad \Gamma \vdash f_- : (\tau_1, \dots, \tau_m) \rightarrow \tau}{\left(\text{EXP}[\tau, \pi] \rightarrow f_- \left(\underset{i=1}{\overset{m}{,}} \text{EXP}[\tau_i, \pi] \right) \right) \in P_\Gamma} \text{INV}_\varphi$$

$$\frac{\text{EXP}[\tau, \pi] \in V_\Gamma \quad \tau = \tau' \quad \tau, \tau' \in \mathbb{T}}{\left(\text{EXP}[\tau, \pi] \rightarrow \text{if EXP}[\mathbb{B}, \pi] \{ \text{EXP}[\tau, \pi] \} \text{else } \{ \text{EXP}[\tau', \pi] \} \right) \in P_\Gamma} \text{IFE}_\varphi$$

$$\frac{\text{EXP}[\hat{\tau}, \pi] \in V_\Gamma \quad \delta_{\text{OPR}}(\odot, \tau, \tau') = \hat{\tau} \quad \odot \in \{==, <, +, *\}}{\left(\text{EXP}[\hat{\tau}, \pi] \rightarrow (\text{EXP}[\tau, \pi] \odot \text{EXP}[\tau', \pi]) \right) \in P_\Gamma} \text{OPX}_\varphi$$

Finally, we normalize to Chomsky Normal Form (CNF), rewriting all productions to either **(1)** ($w \rightarrow xz$) : $V \times V^2$ or **(2)** ($w \rightarrow t$) : $V \times \Sigma$.

Addendum: CFG \cap NFA closure and G_{\cap} construction

Bar-Hillel (1961): For any CFG G , and NFA $A = \langle Q, \Sigma, \delta, q_\alpha, F \rangle$, $\exists G_{\cap}$ s.t. $\mathcal{L}(G_{\cap}) = \mathcal{L}(G) \cap \mathcal{L}(A)$. Salomaa's (1973) construction:

$$\frac{q_\omega \in F}{(S_{\cap} \rightarrow q_\alpha S q_\omega) \in P_{\cap}} S \quad \frac{(W \rightarrow a) \in P \quad (p \xrightarrow{a} r) \in \delta}{(pWr \rightarrow a) \in P_{\cap}} \uparrow$$
$$\frac{(W \rightarrow XZ) \in P \quad p, q, r \in Q}{(pWr \rightarrow (pXq)(qZr)) \in P_{\cap}} \bowtie$$

but, there is a *much* more efficient construction. Intuition: want to show $q_\alpha \rightsquigarrow q_\omega$ in A such that $q_\omega : F$ where $q_\alpha \rightsquigarrow q_\omega \vdash S$. At least one of two cases must hold for $w \in V$ to parse a given $p \rightsquigarrow r$ pair:

1. $\exists a. ((p \xrightarrow{a} r) \in \delta \wedge (w \rightarrow a) \in P)$, or,
2. $\exists q, x, z. ((w \rightarrow xz) \in P \wedge \overbrace{p \rightsquigarrow q, q \rightsquigarrow r}^w)$.

Finite intersection as matrix exponentiation on $(2^V, \oplus, \otimes)$

Let $M \in (2^V)^{|Q| \times |Q|}$, with entries $M[r, c] \subseteq V$ (a set of nonterminals), and let $X \oplus Z = X \cup Z$, $X \otimes Z = \{ w \mid \exists x \in X, z \in Z. (w \rightarrow xz) \in P \}$.

$$M_0[r, c] = \bigcup_{a \in \Sigma} \left\{ w \mid (w \rightarrow a) \in P \wedge (q_r \xrightarrow{a} q_c) \in \delta \right\}.$$

We will define the matrix exponential in the standard manner:

$$\exp(M_0) = \sum_{i=0}^{\infty} M_0^i = \sum_{i=0}^{|Q|} M_0^i \quad (\alpha_\emptyset \Leftrightarrow \text{S.U.T.} \Rightarrow \text{nilpotent}).$$

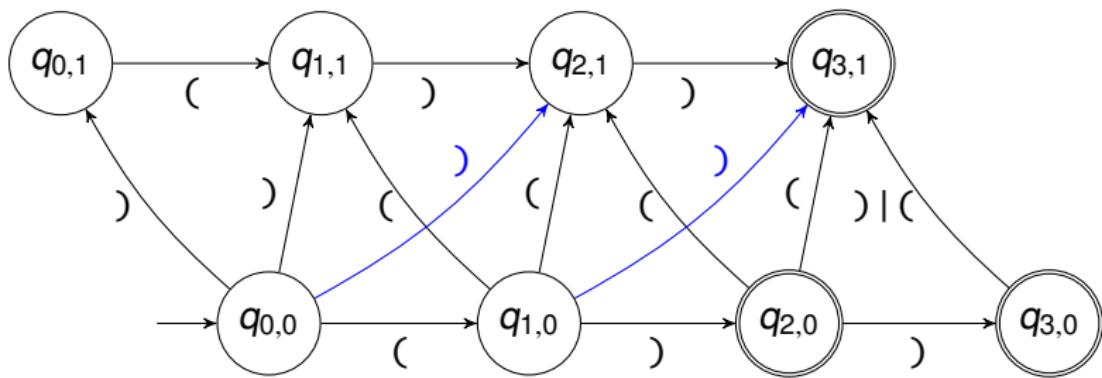
$$T(2n) = \sum_{i=0}^{2n} M_0^i = \begin{cases} M_0, & n = 1, \\ T(n) \oplus (T(n) \cdot T(n)), & \text{otherwise.} \end{cases}$$

The following proposition decides nonemptiness:

$$\left[\bigvee_{q_\omega \in F} S \in \exp(M_0)[q_\alpha, q_\omega] \right] \iff \mathcal{L}(G) \cap \mathcal{L}(\alpha_\cap) \neq \emptyset$$

Repair example: Simple Levenshtein automaton

Suppose we have the string, $\sigma = (\))$ and wish to balance the parentheses. Assume we have the Chomsky Normal Form CFG, $G' = \{S \rightarrow LR, S \rightarrow LF, S \rightarrow SS, F \rightarrow SR, L \rightarrow (, R \rightarrow)\}$ and let us impose an ordering of S, F, L, R on V . We will initially have the Levenshtein automaton, α_\emptyset , depicted below:



n.b. acyclic, therefore has strictly upper triangular adjacency matrix.

Repair example: Initial parse chart (M_0)

M_0	q_{00}	q_{01}	q_{10}	q_{11}	q_{20}	q_{21}	q_{30}	q_{31}
q_{00}	$SFLR$ □□□■	$SFLR$ □□■□	$SFLR$ □□□■	$SFLR$ □□□□	$SFLR$ □□□■	$SFLR$ □□□□	$SFLR$ □□□□	$SFLR$ □□□□
q_{01}		□□□□		□□■□	□□□□	□□□□	□□□□	□□□□
q_{10}				□□■□	□□□■	□□■□	□□□□	□□□■
q_{11}					□□□□	□□□■	□□□□	□□□□
q_{20}						□□■□	□□□■	□□■□
q_{21}							□□□□	□□□■
q_{30}								□□■■
q_{31}								

Initial configuration, after filling all unit productions.

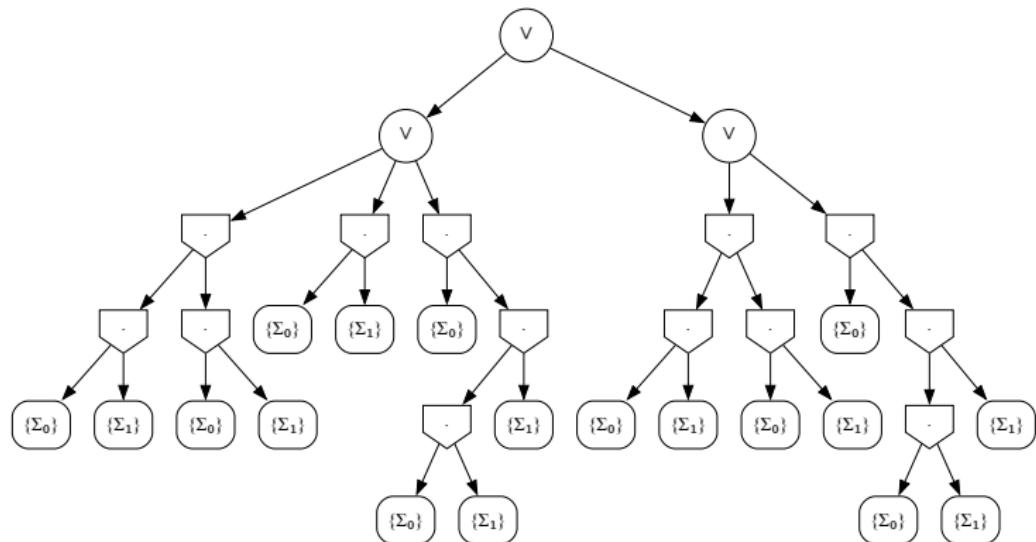
Repair example: Final parse chart (e^{M_0})

M_∞	q_{00}	q_{01}	q_{10}	q_{11}	q_{20}	q_{21}	q_{30}	q_{31}
q_{00}	$SFLR$ □□□■	$SFLR$ □□■□	$SFLR$ □□□■	$SFLR$ ■□□□	$SFLR$ □□□■	$SFLR$ □■□□	$SFLR$ ■□□□	$SFLR$ ■□□□
q_{01}		□□□□		□□■□	□□□□	■□□□	□□□□	□■□□
q_{10}				□□■□	□□□■	■□■□	□□□□	■■□□
q_{11}				□□□□		□□□■	□□□□	□□□□
q_{20}					□□■□	□□□■		■□■□
q_{21}						□□□□	□□□■	
q_{30}							□□■■	
q_{31}								

Final configuration, after matrix fixpoint is reached.

Repair example: Regex denoting $\mathcal{L}(G) \cap \mathcal{L}(\alpha_\emptyset)$

$$\left(\text{a b a b} \mid \left(\text{a b} \mid \text{a a b b} \right) \right) \mid \left(\text{a b a b} \mid \text{a a b b} \right)$$



Regular expression reconstructed from the final parse chart.

Sampling star-free regular expressions uniformly

Let $e : E$ be an SFRE with two connectives: $e \rightarrow \Sigma | e \cdot e | e \vee e$.

Theorem (Uniform tree enumeration)

To sample parse trees, take a PRNG and feed it into `enum`:

$$\text{enum}(e, n) = \begin{cases} e & \text{if } e \in \Sigma \\ \text{enum}\left(x, \lfloor \frac{n}{|z|} \rfloor\right) \cdot \text{enum}\left(z, n \bmod |z|\right) & \text{if } e = x \cdot z \\ \text{enum}\left((x, z)_{\min(1, \lfloor \frac{n}{|x|} \rfloor)}, n - |x| \min(1, \lfloor \frac{n}{|x|} \rfloor)\right) & \text{if } e = x \vee z \end{cases}$$

Where the number of parse trees in a SFRE we abbreviate as $|e|$:

$$|e| : E \rightarrow \mathbb{N} = \begin{cases} 1 & \text{if } e \in \Sigma \\ x \times z & \text{if } e = x \cdot z \\ x + z & \text{if } e = x \vee z \end{cases}$$

n.b. we may need to disambiguate to guarantee $\mathcal{L}(e)$ uniformity.

Sampling star-free regular expressions autoregressively

Now, for any SFRE, e , choose (e) witnesses $\sigma \in \mathcal{L}(e)$:

$$\text{follow}(e) = \begin{cases} \{e\} & \text{if } e \in \Sigma \\ \text{follow}(x) & \text{if } e = x \cdot z \\ \text{follow}(x) \cup \text{follow}(z) & \text{if } e = x \vee z \end{cases}$$

$$\text{choose}(e) = \begin{cases} e & \text{if } e \in \Sigma \\ (s \rightsquigarrow \text{follow}(e)) \cdot \text{choose}(\delta_s e) & \text{if } e = x \cdot z \\ \text{choose}(e' \rightsquigarrow \{x, z\}) & \text{if } e = x \vee z \end{cases}$$

where $\delta_s e$ is the Brzozowskian derivative (1973) and \rightsquigarrow denotes probabilistic choice from a small finite set. This may be augmented with a weighted choice operator, $\sigma \rightsquigarrow P_\theta(\sigma_n | \sigma_{n-1}, \dots, \sigma_{n-k})$.

Evaluation benchmarks

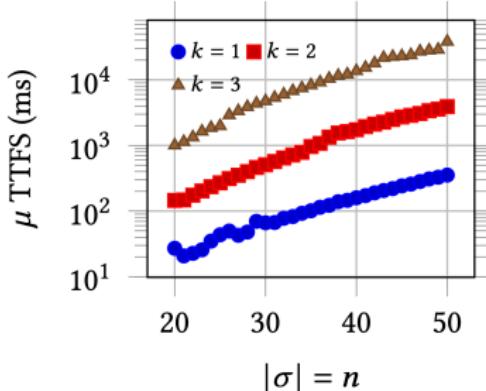
Experimental Setup

- ▶ Arity: $k \in \{1, 2, 3\}$
Fixed: $|\Gamma| = 18$, $|\mathbb{T}| = 7$
- ▶ CNF grammar sizes:
 $|G'_\Gamma| \in [1.9 \times 10^4, 9.9 \times 10^5]$
- ▶ Apple M4 (16 GB RAM)

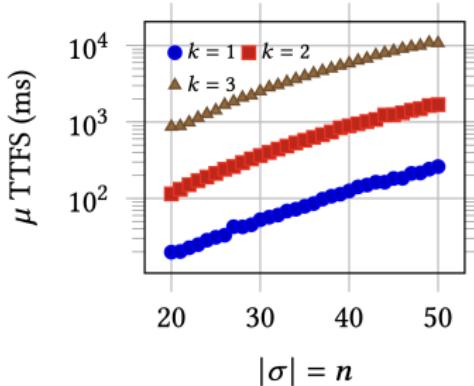
Benchmarks

- ▶ **Slicing:** $\sigma \leftarrow \mathcal{L}(G'_\Gamma) \cap \Sigma^n$
- ▶ **Type inference:** reuse random functions from slice sampling,
replace $(:\tau=)$ with $(:\Sigma=)$, and
 $\sigma' \leftarrow \mathcal{L}(G'_\Gamma) \cap (\dots : \Sigma = \dots)$
- ▶ **Bounded delay:** 1786 ± 817 ns
- ▶ **Throughput:** $\sim 2.2 \times 10^7$ tok/s

Slice sampling delay



Type inference delay



Future work

- ▶ More compact embeddings and asymptotic complexity
- ▶ Laziness: instantiate CFG productions during parsing
- ▶ Extend to richer type systems, e.g., polymorphism, higher-order functions, subtyping, nested scope
- ▶ “A Relevance Sampler for $\mu\text{Rust}_{\text{SL}}$ ” (Considine, 2025) explores a straight-line fragment of Rust with nominal relevance
- ▶ “A Tree Sampler for Bounded CFLs” (Considine, 2024) describes a uniform sampler for finite CFL intersections
- ▶ Other FPT embeddings. Open to suggestions!
- ▶ Applications to program synthesis and repair
- ▶ Try it yourself at: <https://tidyparse.github.io>

Acknowledgements

Thank you!

- ▶ Ori Roth
- ▶ Chuta Sano
- ▶ Brigitte Pientka
- ▶ David Bieber
- ▶ Margaret Considine
- ▶ Mark Considine

Lastly, thank you to the LAFI organizers!