

# A Word Sampler for Well-Typed Functions

Breandan Mark Considine

January 10, 2026

# Formal languages & type theory

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

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

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

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

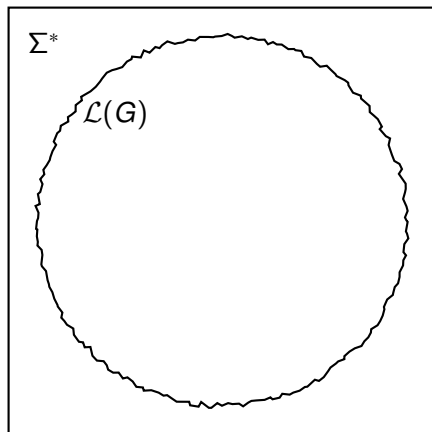
# Programming language [in]approximability

- ▶  $\Sigma^*$ : all words over  $\Sigma$

$\Sigma^*$

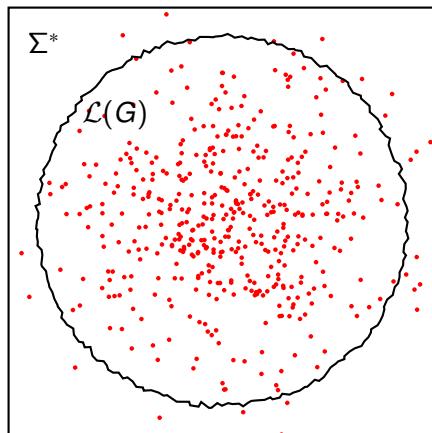
# Programming language [in]approximability

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



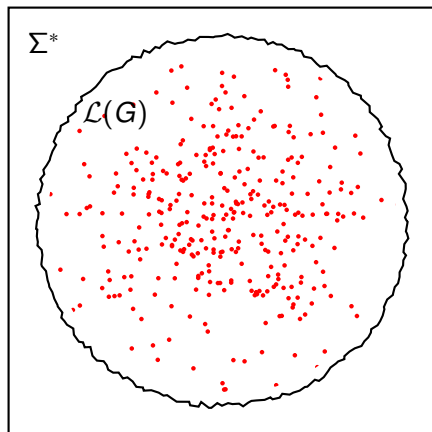
# Programming language [in]approximability

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



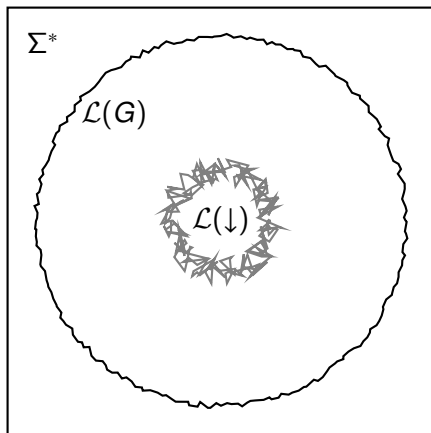
# Programming language [in]approximability

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



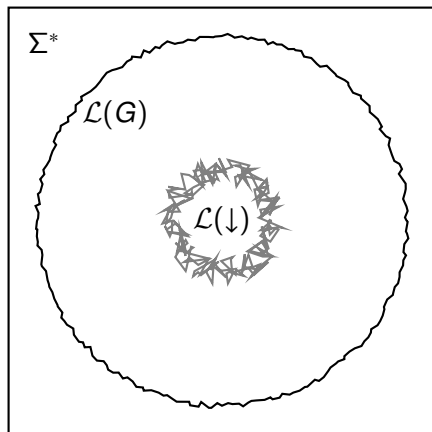
# Programming language [in]approximability

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



# Programming language [in]approximability

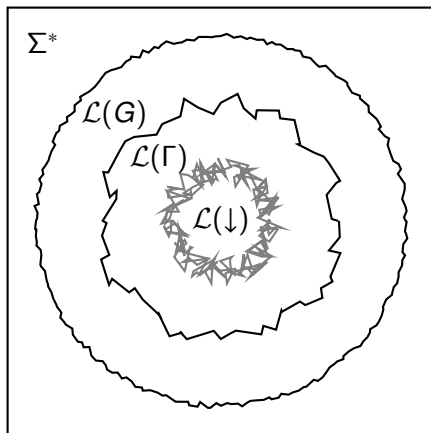
- ▶  $\Sigma^*$ : all words over  $\Sigma$
- ▶  $\mathcal{L}(G)$ : syntactically valid
- ▶ Most LLMs:  $\sigma \leftarrow \Sigma^*$
- ▶ Guidance:  $\sigma \leftarrow \mathcal{L}(G)$
- ▶  $\mathcal{L}(\downarrow)$ : halting programs
- ▶ Tighter approximations require ever-increasing expressive power





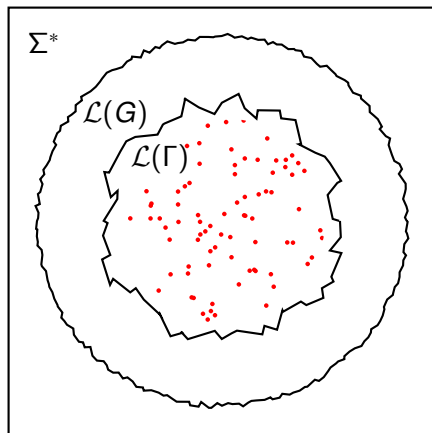
# Programming language [in]approximability

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



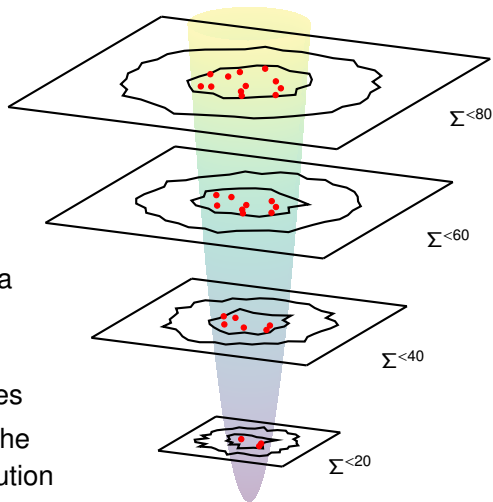
# Programming language [in]approximability

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



# 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  $\Gamma$
- ▶ 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  ::=  $\ulcorner N \urcorner$  |  $\ulcorner B \urcorner$  | 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
 $\ulcorner B \urcorner$  ::= true | false
 $\ulcorner N \urcorner$  ::= 1 | 2 | 3 | ...
```

**Type universe:** Finite  $\mathbb{T}$  with two primitive types (e.g.,  $\mathbb{B}, \mathbb{N}, \dots$ )

**Ambient context:**  $\Gamma$  maps  $f_- : (\tau_1, \dots, \tau_m) \rightarrow \tau$ .

## Expression fragment: static semantics

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

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

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

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

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

# 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.

$$\frac{\langle \vec{\tau}, \dot{\tau} \rangle \in \mathbb{T}^{0..k} \times \mathbb{T} \quad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left( S_\Gamma \rightarrow \text{fun } \text{f0} \left( \bigg[ \begin{smallmatrix} |\vec{\tau}| \\ \text{,} \end{smallmatrix} \right] \left( p_i : \vec{\tau}_i \right) \right) : \dot{\tau} = \text{EXP}[\dot{\tau}, \vec{\tau} \rightarrow \dot{\tau}] \right) \in P_\Gamma} \text{FUN}_\varphi$$

$$\frac{\text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \in V_\Gamma \quad \tau = \dot{\tau} \quad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left( \text{EXP}[\tau, \vec{\tau} \rightarrow \dot{\tau}] \rightarrow \text{f0} \left( \bigg[ \begin{smallmatrix} |\vec{\tau}| \\ \text{,} \end{smallmatrix} \right] \text{EXP}[\vec{\tau}_i, \vec{\tau} \rightarrow \dot{\tau}] \right) \right) \in P_\Gamma} \text{REC}_\varphi$$

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

## Embedding the type checker (II)

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

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

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

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:

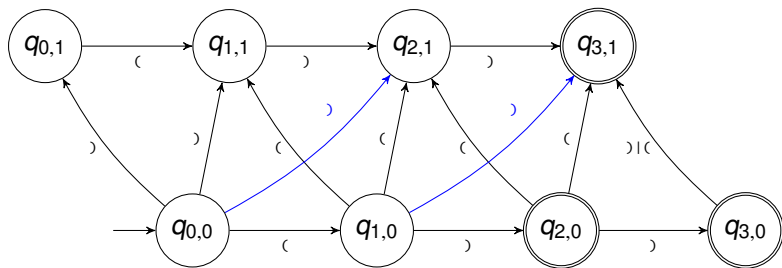
$$\frac{q_\omega \in F}{(S_\cap \rightarrow q_\alpha S q_\omega) \in P_\cap} \mathcal{S} \quad \frac{(W \rightarrow a) \in P \quad (p \xrightarrow{a} r) \in \delta}{(pWr \rightarrow a) \in P_\cap} \uparrow$$
$$\frac{(W \rightarrow XZ) \in P \quad p, q, r \in Q}{(pWr \rightarrow (pXq)(qZr)) \in P_\cap}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

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.  $\exists q, x, z. ((w \rightarrow xz) \in P \wedge \underbrace{p \rightsquigarrow q}_x \overbrace{q \rightsquigarrow r}^z)$ .

## 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,  $A$ , depicted below.



# Repair example: Initial parse chart configuration

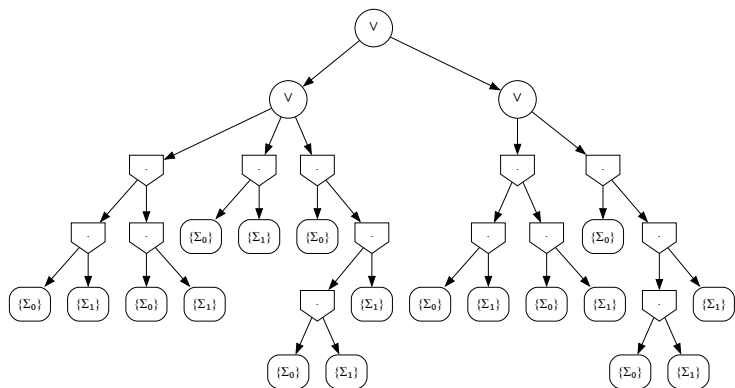
$M_0$	$q_{00}$	$q_{01}$	$q_{10}$	$q_{11}$	$q_{20}$	$q_{21}$	$q_{30}$	$q_{31}$
$q_{00}$		SFLR □□□■	SFLR □□■□	SFLR □□□■	SFLR □□□□	SFLR □□□■	SFLR □□□□	SFLR □□□□
$q_{01}$			□□□□	□□■□	□□□□	□□□□	□□□□	□□□□
$q_{10}$				□□■□	□□□■	□□■□	□□□□	□□□■
$q_{11}$					□□□□	□□□■	□□□□	□□□□
$q_{20}$						□□■□	□□□■	□□■□
$q_{21}$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

# Repair example: Final parse chart configuration

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

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

$(abab \mid (ab \mid aabb)) \mid (abab \mid aabb)$



# Sampling star-free regular expressions uniformly

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

## 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 SFGRE,  $e$ , choose  $(e)$  witnesses  $\sigma \in \mathcal{L}(e)$ :

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

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

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