Bounded Resources as Languages:
A Grammatical Embedding of Substructural Constraints

Breandan Mark Considine

January 27, 2026

Language theory and type theory

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

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

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

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

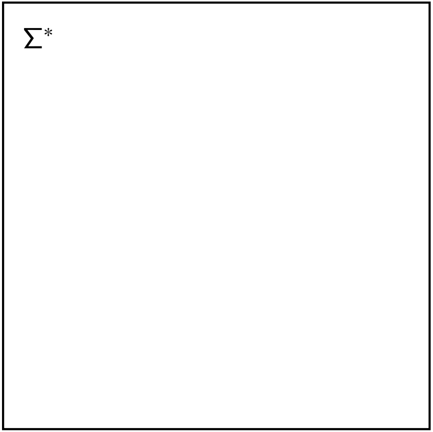
Linear logic (LL) and language theory (LT)

	LL	LT	Interpretation
Conjunction (Multiplicative)	$A \otimes B$	$A \cdot B$	Concatenation[1] $\{a \cdot b \mid a \in \mathcal{L}_A \wedge b \in \mathcal{L}_B\}$
Unit	1	ε	Empty string
Disjunction (Additive)	$A \oplus B$	$A \vee B$	Union $\mathcal{L}_A \cup \mathcal{L}_B$
Conjunction (Additive)	$A \& B$	$A \wedge B$	Intersection $\mathcal{L}_A \cap \mathcal{L}_B$
Iteration (Exponential)	$!A$	A^*	Kleene Star $\mathcal{L}(A^0 \cup A^1 \cup A^2 \cup \cdots)$
Implication (Residual)	$A \multimap B$	$A \backslash B$	Left Quotient $\{b \mid \mathcal{L}_A \cdot b \cap \mathcal{L}_B \neq \varnothing\}$

[1] *n.b.: We do not assume commutativity* $(A \otimes B \neq B \otimes A)$ *in formal languages.*

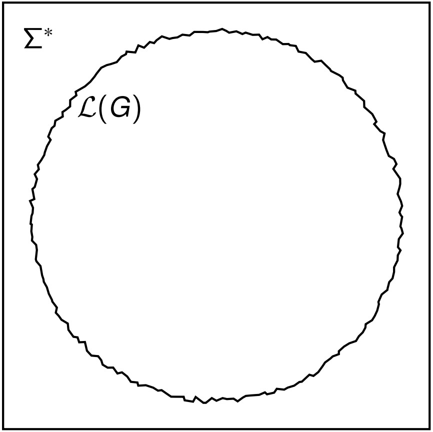
Programming language [in]approximability

- Σ^*: all words over Σ



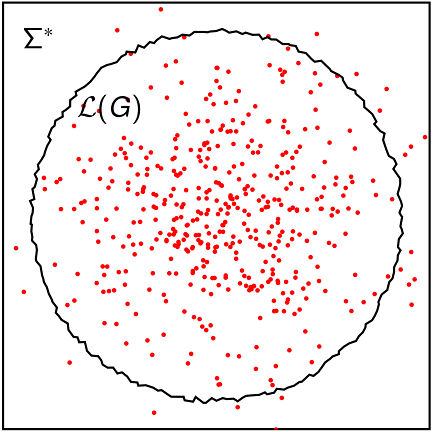
Programming language [in]approximability

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



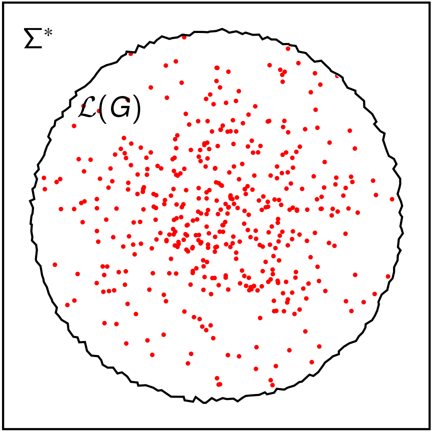
Programming language [in]approximability

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



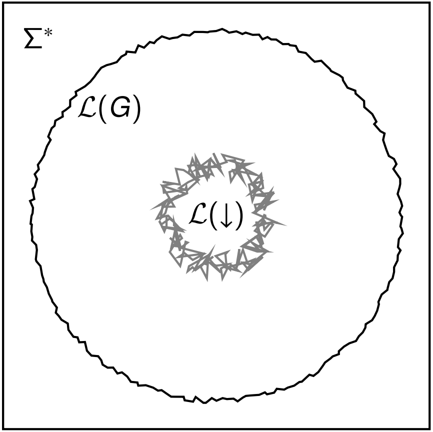
Programming language [in]approximability

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



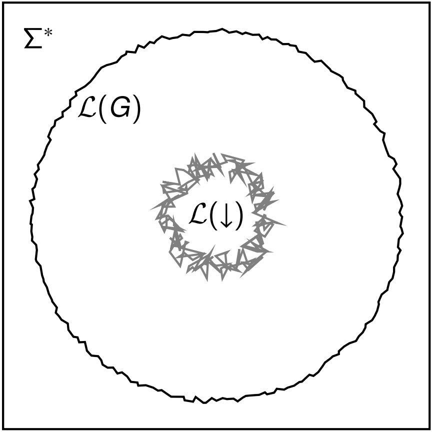
Programming language [in]approximability

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



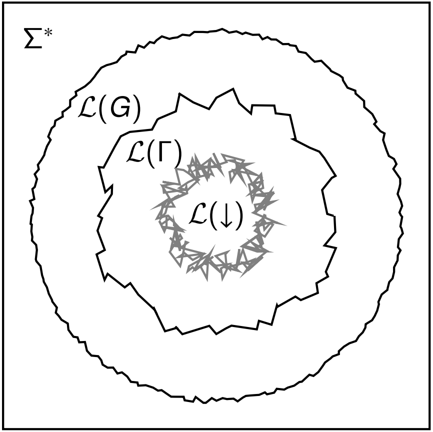
Programming language [in]approximability

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



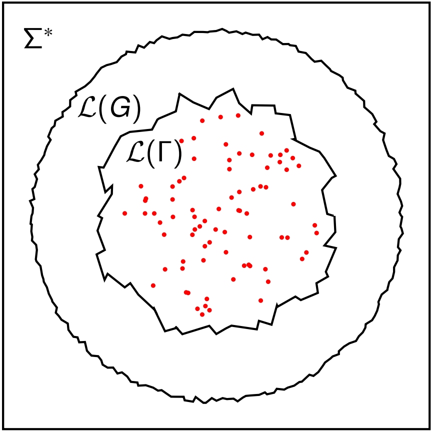
Programming language [in]approximability

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



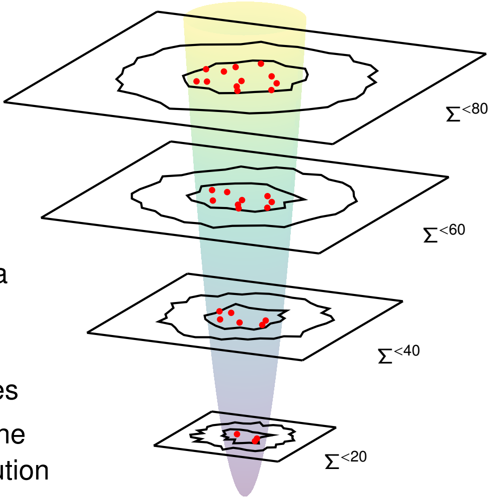
Programming language [in]approximability

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



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, $\mathtt{E}[\tau]$
- Each typing judgment becomes a schema for constructing a family of synthetic productions, each instantiated with $\tau : \mathbb{T}$
- Syntax decorators, $\Phi_{p,\tau} : (P \times \mathbb{T}) \to \mathbb{T}^+ \to (V \cup \Sigma)^+$

Syntax: $$\frac{\Gamma \vdash \mathtt{E}_i : \tau_i \qquad p = \left(\mathtt{E} \to (\Sigma^* \; \mathtt{E}_i \; \Sigma^*)^{m_{\geq 1}}\right) \in P}{\left(\mathtt{E}[\tau] \; \to \; \Phi_{p,\tau}(\tau_1, \; \ldots, \tau_m)\right) \in P_\Gamma}$$

Names: $$\Gamma \vdash e : \tau \;\; \Rightarrow \;\; \left(\mathtt{E}[\tau] \; \to \; \mathtt{e}\right) \in P_\Gamma$$

Functions: $$\frac{\Gamma \vdash f : (\tau_1, \; \ldots, \tau_k) \to \tau}{\left(\mathtt{E}[\tau_1] \; \to \; \mathtt{f} \;\; \mathtt{(} \;\; \mathtt{E}[\tau_1] \;\; \mathtt{,} \;\; \ldots \;\; \mathtt{,} \;\; \mathtt{E}[\tau_k] \;\; \mathtt{)}\right) \in P_\Gamma}$$

Example language: simply typed function syntax

$$
\begin{array}{lcl}
\texttt{FUN} & ::= & \texttt{fun}\ \texttt{f0}\ (\ \texttt{PRM}\ )\ :\ \mathbb{T}\ =\ \texttt{EXP} \\
\texttt{PRM} & ::= & \texttt{PID}\ :\ \mathbb{T} \mid \texttt{PRM}\ ,\ \texttt{PID}\ :\ \mathbb{T} \\
\texttt{EXP} & ::= & \ulcorner\mathbb{N}\lrcorner \mid \ulcorner\mathbb{B}\lrcorner \mid \texttt{PID} \mid \texttt{INV} \mid \texttt{IFE} \mid \texttt{OPX} \\
\texttt{OPX} & ::= & (\ \texttt{EXP}\ \texttt{OPR}\ \texttt{EXP}\ ) \\
\texttt{IFE} & ::= & \texttt{if}\ \texttt{EXP}\ \{\ \texttt{EXP}\ \}\ \texttt{else}\ \{\ \texttt{EXP}\ \} \\
\texttt{INV} & ::= & \texttt{FID}\ (\ \texttt{ARG}\ ) \\
\texttt{ARG} & ::= & \texttt{EXP} \mid \texttt{ARG}\ ,\ \texttt{EXP} \\
\texttt{OPR} & ::= & + \mid * \mid < \mid == \\
\texttt{PID} & ::= & \texttt{p1} \mid \ldots \mid \texttt{pk} \\
\texttt{FID} & ::= & \texttt{f0} \mid \texttt{f1} \mid \ldots \mid \texttt{fn} \\
\ulcorner\mathbb{B}\lrcorner & ::= & \texttt{true} \mid \texttt{false} \\
\ulcorner\mathbb{N}\lrcorner & ::= & 1 \mid 2 \mid 3 \mid \ldots
\end{array}
$$

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

Ambient context: Γ maps $\texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau$.

Expression fragment: static semantics

$$\frac{\Gamma \vdash e_c : \mathbb{B} \qquad \Gamma \vdash e_\top : \tau \qquad \Gamma \vdash e_\bot : \tau}{\Gamma \vdash \texttt{if}\ e_c\ \texttt{\{}\ e_\top\ \texttt{\}}\ \texttt{else}\ \texttt{\{}\ e_\bot\ \texttt{\}} : \tau}\ \texttt{IFE}$$

$$\frac{\Gamma \vdash \texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau \qquad \Gamma \vdash e_i : \tau_i \ \ \forall i \in [1, m]}{\Gamma \vdash \texttt{f\_}\ \texttt{(}\ e_1\ \texttt{,}\ \ldots\ \texttt{,}\ e_m\ \texttt{)} : \tau}\ \texttt{INV}$$

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

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

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

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: $\mathtt{EXP}[\tau, \pi] \qquad \left(\tau \in \mathbb{T},\ \pi \equiv (\vec{\tau} \to \dot{\tau})\right)$

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

$$\frac{\langle \vec{\tau}, \dot{\tau} \rangle \in \mathbb{T}^{0..k} \times \mathbb{T} \qquad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(S_\Gamma \to \mathtt{fun}\ \mathtt{f0}\ \mathtt{(}\ \overset{|\vec{\tau}|}{\underset{i=1}{\boldsymbol{,}}} \left(p_i\ \mathtt{:}\ \vec{\tau}_i\right)\ \mathtt{)}\ \mathtt{:}\ \dot{\tau}\ \mathtt{=}\ \mathtt{EXP}[\dot{\tau}, \vec{\tau} \to \dot{\tau}]\right) \in P_\Gamma}\ \mathtt{FUN}_\varphi$$

$$\frac{\mathtt{EXP}[\tau, \vec{\tau} \to \dot{\tau}] \in V_\Gamma \qquad \tau = \dot{\tau} \qquad \vec{\tau}_{0..|\vec{\tau}|} \in \vec{\tau}}{\left(\mathtt{EXP}[\tau, \vec{\tau} \to \dot{\tau}] \to \mathtt{f0}\ \mathtt{(}\ \overset{|\vec{\tau}|}{\underset{i=1}{\boldsymbol{,}}} \mathtt{EXP}[\vec{\tau}_i, \vec{\tau} \to \dot{\tau}]\ \mathtt{)}\right) \in P_\Gamma}\ \mathtt{REC}_\varphi$$

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

Embedding the type checker (II)

$$\frac{\texttt{EXP}[\tau,\pi] \in V_\Gamma \qquad \Gamma \vdash \texttt{f\_} : (\tau_1,\ldots,\tau_m) \to \tau}{\left(\texttt{EXP}[\tau,\pi] \to \texttt{f\_}\ \texttt{(}\ \overset{m}{\underset{i=1}{\boldsymbol{,}}}\ \texttt{EXP}[\tau_i,\pi]\ \texttt{)}\right) \in P_\Gamma}\ \texttt{INV}_\varphi$$

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

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

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

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:

$$\frac{q_\omega \in F}{\big(S_\cap \to q_\alpha\, S\, q_\omega\big) \in P_\cap}\,\mathcal{S} \qquad \frac{(W \to a) \in P \qquad (p \overset{a}{\to} r) \in \delta}{\big(pWr \to a\big) \in P_\cap}\uparrow$$

$$\frac{(W \to XZ) \in P \qquad p, q, r \in Q}{\big(pWr \to (pXq)\,(qZr)\big) \in P_\cap}\bowtie$$

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. $\exists a.\big((p \overset{a}{\to} r) \in \delta \wedge (w \to a) \in P\big)$, or,
2. $\exists q, x, z.\Big((w \to xz) \in P \wedge \overbrace{\underbrace{p \rightsquigarrow q}_{x}, \underbrace{q \rightsquigarrow r}_{z}}^{w}\Big)$.

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 $X \oplus Z = X \cup Z, X \otimes Z = \left\{ w \,\middle|\, \exists x \in X,\ z \in Z.\ (w \to xz) \in P \right\}$.

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

We will define the matrix exponential in the standard manner:

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

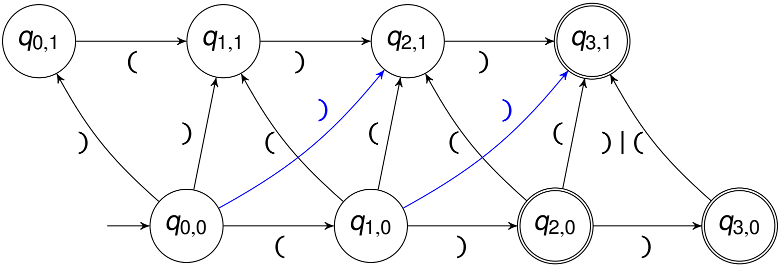
$$T(2n) = \sum_{i=0}^{2n} M_0^i = \begin{cases} M_0, & n = 1, \\ T(n) \ \oplus \ \big(T(n) \cdot T(n)\big), & \text{otherwise.} \end{cases}$$

The following proposition decides nonemptiness:

$$\left[\bigvee_{q_\omega \in F} S \in e^{M_0}[q_\alpha, q_\omega] \right] \Longleftrightarrow \mathcal{L}(G) \cap \mathcal{L}(\alpha_\cap) \neq \varnothing$$

Repair example: Simple Levenshtein automaton

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



n.b. acyclic, therefor 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}
		S F L R	*S F L R*	*S F L R*	*S F L R*	*S F L R*	*S F L R*	*S F L R*
q_{00}		□□□■	□□■□	□□□■	□□□□	□□□■	□□□□	□□□□
q_{01}			□□□□	□□■□	□□□□	□□□□	□□□□	□□□□
q_{10}				□□■□	□□□■	□□■□	□□□□	□□□■
q_{11}					□□□□	□□□■	□□□□	□□□□
q_{20}						□□■□	□□□■	□□■□
q_{21}							□□□□	□□□■
q_{30}								□□■■
q_{31}								

Initial configuration, after filling all unit productions.

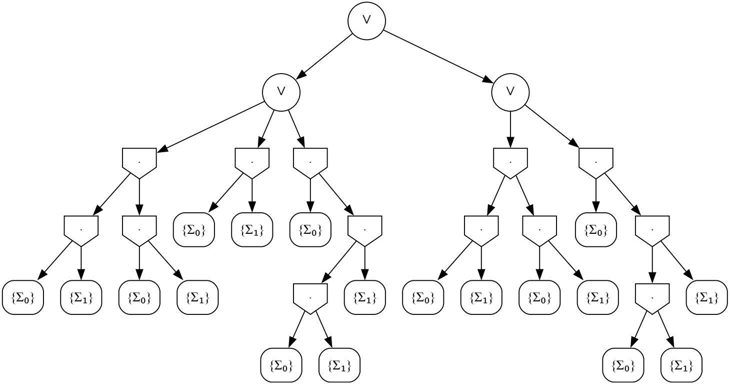
Repair example: Final parse chart (e^{M_0})

M_∞	q_{00}	q_{01}	q_{10}	q_{11}	q_{20}	q_{21}	q_{30}	q_{31}
		$S\ F\ L\ R$	$S\ F\ L\ R$	$S\ F\ L\ R$	$S\ F\ L\ R$	$S\ F\ L\ R$	$S\ F\ L\ R$	$S\ F\ L\ R$
q_{00}		□□□■	□□■□	□□□■	■□□□	□□□■	□■□□	■□□□
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 $\mathcal{L}(G) \cap \mathcal{L}(\alpha_\varnothing)$

$$\Big(\texttt{a b a b} \mid \big(\texttt{a b} \mid \texttt{a a b b}\big)\Big) \mid \big(\texttt{a b a b} \mid \texttt{a a b b}\big)$$



Regular expression reconstructed from the final parse chart.

Enumerative tree sampling

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

Theorem (Uniform tree enumeration)

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

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

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

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

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

Autoregressive Brzozowski sampling

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

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

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

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

Boltzmann Sampling I: From Grammar to Equations

Symbolic Method

First map the structural specification (i.e., the CFG) to a system of equations $\mathbf{y}(x) = \Phi(\mathbf{y}(x), x)$.

Translation:

- **Union** $(\mathcal{A} \cong \mathcal{B} + \mathcal{C}) \to$ Sum $\big(A(x) = B(x) + C(x)\big)$
- **Product** $(\mathcal{A} \cong \mathcal{B} \times \mathcal{C}) \to$ Product $\big(A(x) = B(x) \cdot C(x)\big)$
- **Atom** $(\mathcal{A} \cong \mathcal{Z}) \to$ Variable $\big(A(x) = x\big)$

Example

A system with non-terminals $\mathtt{U}, \mathtt{V}$ yields $\mathbf{y} = \big[U(x), V(x)\big]^T$:

$$\begin{array}{l} \mathtt{U} \to \mathtt{a}\ \mathtt{V}\ \mathtt{U} \mid \mathtt{b}\ \mathtt{V} \mid \mathtt{c} \\ \mathtt{V} \to \mathtt{d}\ \mathtt{U}\ \mathtt{U} \mid \mathtt{e} \end{array} \quad \Rightarrow \quad \begin{cases} U(x) = xV(x)U(x) + xV(x) + x \\ V(x) = xU(x)^2 + x \end{cases}$$

Boltzmann Sampling II: Tuning the Mean Size

Objective: For nonterminal C, target mean size n by tuning x:

$$\mathbb{E}_x[\text{Size}] = \frac{x \cdot C'(x)}{C(x)} = n$$

Newton Iteration: Find oracle weights $\mathbf{y}$ by solving:

$$\mathbf{F}(\mathbf{y}) = \mathbf{y} - \Phi(\mathbf{y}, x) = \mathbf{0}$$

- **Jacobian:** Compute $\mathbf{J} = \mathbf{I} - \frac{\partial \Phi}{\partial \mathbf{y}}$ (e.g., via AD, SD, or FD).
- **Update step:** Iterate until convergence:

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \mathbf{J}^{-1}\mathbf{F}(\mathbf{y}_k)$$

This will converge quadratically and we obtain $C(x)$ and $C'(x)$.

Boltzmann Sampling III: Recursive Generation

A Boltzmann sampler $\Gamma C(x)$ draws $\gamma \in \mathcal{C}$ with $\mathbb{P}_x(\gamma) = x^{|\gamma|}/C(x)$
i.e., roll a weighted die, pick a branch according to $\mathbf{y}$.

Algorithm (class $\mathcal{A} = \mathcal{B}_1 + \mathcal{B}_2 + \dots$):

1. **Local Weights:** Retrieve value $A(x)$ from $\mathbf{y}$. The probability of choosing rule k is its share of the total weight:

$$\pi_k = \frac{B_k(x)}{A(x)}$$

2. **Choice:** Pick rule k with probability π_k.
3. **Recurse:** If rule k is a product, e.g., $\mathcal{C} \times \mathcal{D}$, each component $\mathcal{C}$ and $\mathcal{D}$ can be generated independently.

Note: Pre-calculating $\mathbf{y}(x)$ *ensures* $\mathcal{O}(1)$ *cost per node generated.*

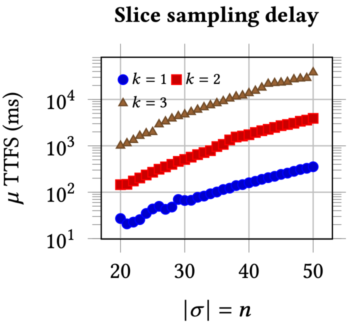
Evaluation benchmarks

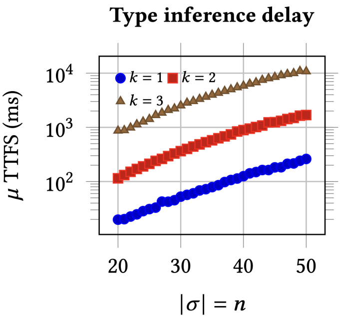
Experimental Setup

- 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)

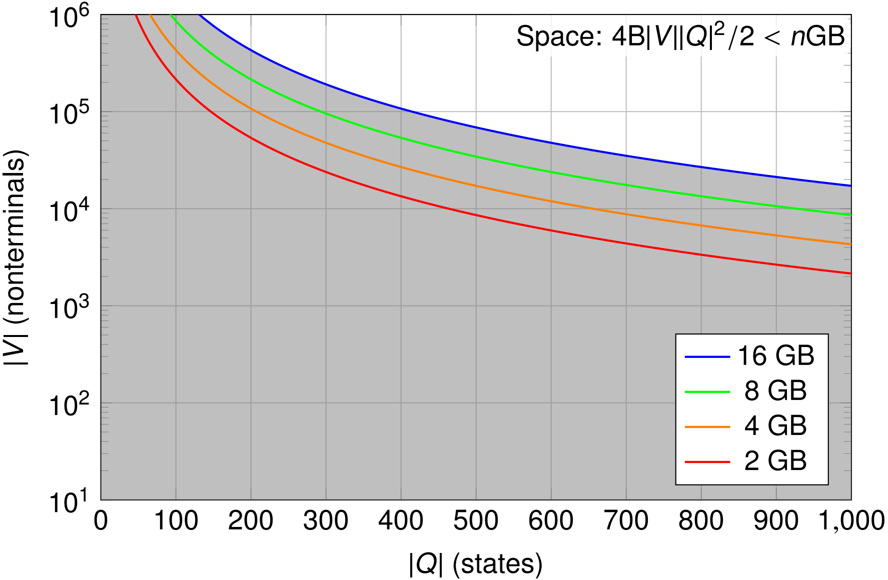
Benchmarks

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





$\mathcal{L}(G) \cap \mathcal{L}(\alpha_\varnothing)$ instances feasible in under 16 GB



μRust: Syntax and semantics

Consider a singly-typed language with the following terms:

$$
\begin{array}{lcl}
\mathtt{FUN} & ::= & \mathtt{fn}\ \mathtt{f0}\ \mathtt{(}\ \mathtt{PRM}\ \mathtt{)}\ \mathtt{->}\ \mathbb{T}\ \mathtt{\{}\ \mathtt{BDY}\ \mathtt{\}} \\
\mathtt{PRM} & ::= & \mathtt{PID}\ \mathtt{:}\ \mathbb{T} \mid \mathtt{PRM}\ \mathtt{,}\ \mathtt{PID}\ \mathtt{:}\ \mathbb{T} \\
\mathtt{BDY} & ::= & \mathtt{INV} \mid \mathtt{STM}\ \mathtt{BDY} \\
\mathtt{STM} & ::= & \mathtt{let}\ \mathtt{PID}\ \mathtt{=}\ \mathtt{INV}\ \mathtt{;} \\
\mathtt{INV} & ::= & \mathtt{FID}\ \mathtt{(}\ \mathtt{ARG}\ \mathtt{)} \\
\mathtt{ARG} & ::= & \mathtt{PID} \mid \mathtt{ARG}\ \mathtt{,}\ \mathtt{ARG} \\
\mathtt{PID} & ::= & \mathtt{p1} \mid \ldots \mid \mathtt{pk} \\
\mathtt{FID} & ::= & \mathtt{f0} \mid \mathtt{f1} \mid \ldots \mid \mathtt{fm}
\end{array}
$$

Assume an ambient context, Γ, consisting of $\mathtt{f1}, \ldots, \mathtt{fm}$:

$$
\Gamma \quad ::= \quad \varnothing \quad | \quad \Gamma,\ \mathtt{f\_} : (\tau_1, \ldots, \tau_k) \to \tau
$$

The unrestricted semantics are conventional.

μRust: Examples

This admits straight line programs (SLPs) of the following shape,

```
fn f0(p1 : T, p2 : T) -> T {
    let p3 = mul(p1, p2);
    let p4 = add(p1, p1);
    let p5 = mul(p1, p3);
    let p6 = add(p3, p1);
    add(p3, p5)
}
```

```
> Warning: p4, p6 are unused!
```

however unused resources, i.e., names may remain after returning.

Ambient context

Now, let us interpolate f0 as a string inside the following context:

```
[forbid(unused_variables)]
[derive(Clone, Copy, Debug)]
[must_use]
pub struct T(i128);
fn add(_ : T, _ : T) -> T { T(0) }
fn mul(_ : T, _ : T) -> T { T(0) }
...
fn f0(p1 : T, ..., pk : T) -> T { <...> }
```

μRust$_{\text{SL}}$: Relevance semantics

Obligations. For `f0` with parameters $p_1 : \tau_1, \ldots, p_k : \tau_k$, initialize $\Phi = \{p_1, \ldots, p_k\}$. Each bound name must be used *at least once*. Locals introduced by `let` also carry obligations. Body is well-typed iff all obligations are discharged, i.e., $\Gamma, \Delta \vdash \mathtt{BDY} : \tau \mid \Phi \Rightarrow \varnothing$.

$$\text{Judgments:} \quad \Gamma, \Delta \;\vdash\; t : \tau \mid \Phi \Rightarrow \Phi'.$$

$$\frac{p : \tau \in \Gamma \qquad p \in \Phi}{\Gamma, \Delta \vdash p : \tau \mid \Phi \Rightarrow \Phi \setminus \{p\}}\ (\text{VAR}) \qquad \frac{p : \tau \in \Gamma \qquad p \notin \Phi}{\Gamma, \Delta \vdash p : \tau \mid \Phi \Rightarrow \Phi}\ (\text{VAR}_{\notin})$$

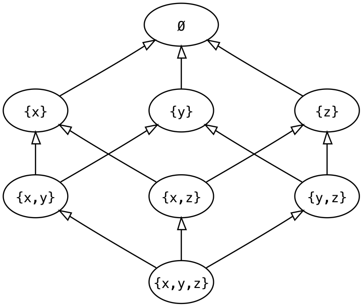
$$\frac{\Gamma \vdash \mathtt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau \qquad \Gamma, \Delta \vdash e_i : \tau_i \mid \Phi_{i-1} \Rightarrow \Phi_i \ \forall i \in [1, m]}{\Gamma, \Delta \vdash \mathtt{f\_}\ \mathtt{(}\ e_1\ \mathtt{,}\ \ldots \mathtt{,}\ e_m\ \mathtt{)} : \tau \mid \Phi_0 \Rightarrow \Phi_m}\ (\text{INV})$$

$$\frac{\Gamma, \Delta \vdash s_1 : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_1 \qquad \Gamma, \Delta \vdash s_2 : \mathtt{unit} \mid \Phi_1 \Rightarrow \Phi_2}{\Gamma, \Delta \vdash s_1\ \mathtt{;}\ s_2 : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_2}\ (\text{SEQ})$$

$$\frac{\Gamma, \Delta \vdash e : \tau \mid \Phi_0 \Rightarrow \Phi_1 \qquad \Gamma, \Delta \vdash x : \tau \mid \Phi_1 \cup \{x\} \Rightarrow \Phi_2}{\Gamma, \Delta \vdash \mathtt{let}\ x\ \mathtt{=}\ e : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_2}\ (\text{LET})$$

CFG embedding: intution

To express all $\Phi \Rightarrow \Phi'$ possible transitions, we must construct a Hasse digram, H_k, e.g., for $k = 3$ parameters $\{x, y, z\}$,



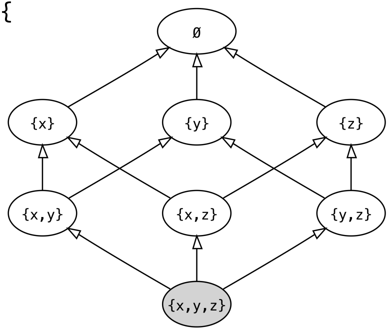
for all relevant productions. This will be tractable for $k \lesssim 10$.

$$\left|\{v, v' \in H_k \mid v \subset v'\}\right| = \sum_{i=0}^{k} \binom{k}{i} \left(2^{k-i} - 1\right) = 3^k - 2^k.$$

Path enumeration

Our grammar will need to express all possible transition paths, e.g.,

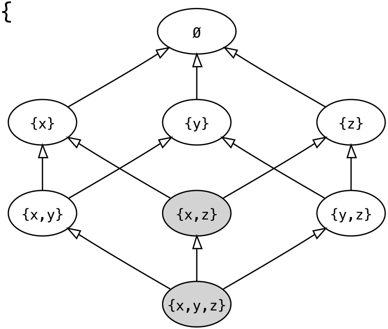
```
fn f0(x : T, y : T, z : T) -> T {
    // Unused: {x,y,z}
}
```



Path enumeration

Our grammar will need to express all possible transition paths, e.g.,

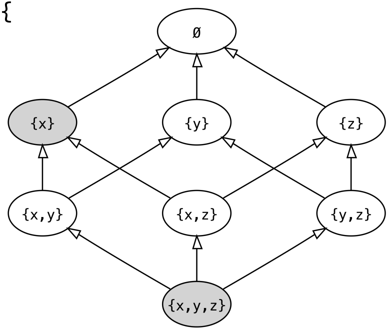
```
fn f0(x : T, y : T, z : T) -> T {
    let _ = mul(y, y); // {x,z}
    // BDY | {x,y,z} => {x,z}
}
```



Path enumeration

Our grammar will need to express all possible transition paths, e.g.,

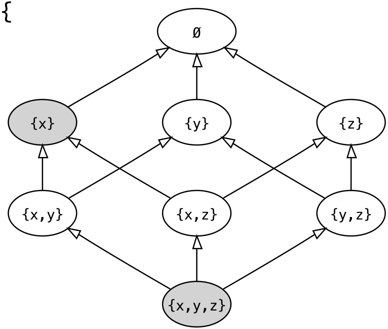
```
fn f0(x : T, y : T, z : T) -> T {
    let _ = mul(y, y); // {x,z}
    let _ = add(y, z); // {x}
    // BDY | {x,y,z}=>{x}
}
```



Path enumeration

Our grammar will need to express all possible transition paths, e.g.,

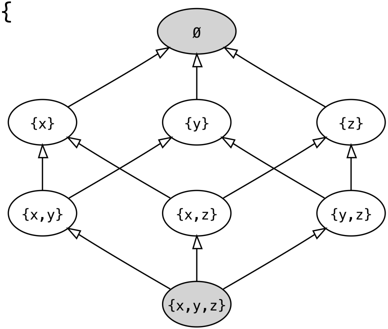
```
fn f0(x : T, y : T, z : T) -> T {
    let _ = mul(y, y); // {x,z}
    let _ = add(y, z); // {x}
    let _ = mul(z, y); // {x}
    // BDY | {x,y,z}=>{x}
}
```



Path enumeration

Our grammar will need to express all possible transition paths, e.g.,

```
fn f0(x : T, y : T, z : T) -> T {
    let _ = mul(y, y); // {x,z}
    let _ = add(y, z); // {x}
    let _ = mul(z, y); // {x}
    let _ = mul(x, z); // {}
    // BDY | {x,y,z}=>{}
}
```



CFG embedding

We want to permit only functions with no outstanding obligations. Construct a CFG, $G_\Gamma = \langle \Sigma, V_\Gamma, P_\Gamma, S_\Gamma \rangle$:

$$\frac{\vec{\tau} : \mathbb{T}^{0..k} \qquad \Phi \Rightarrow \Phi' = \{p_i, \ldots, p_k\} \Rightarrow \varnothing}{\left(S_\Gamma \to \texttt{fn}\ \texttt{f0}\ \texttt{(} \mathop{\text{,}}_{i=1}^{|\vec{\tau}|} \left(p_i\ \texttt{:}\ \vec{\tau}_i\right) \texttt{)}\ \texttt{:}\ \tau\ \texttt{=}\ \texttt{BDY}[\tau, \Phi \Rightarrow \Phi']\right) \in P_\Gamma}\ \texttt{FUN}_\varphi$$

We will decorate nonterminals with a pair of (1) the expression's local return type (τ), and (2) relevance obligations ($\Phi \Rightarrow \Phi'$):

$$\frac{\Gamma \vdash \texttt{f\_} : (\tau_1, \ldots, \tau_m) \to \tau \qquad \Phi' \subseteq \Phi \qquad \Phi \setminus \Phi' = \bigcup_{i=1}^{m} \{p_i\}}{\left(\texttt{INV}[\tau, \Phi \Rightarrow \Phi']\ \to\ \texttt{f\_}\ \texttt{(} \mathop{\text{,}}_{i=1}^{m} p_i\ \texttt{)}\right) \in P_\Gamma}\ \texttt{INV}_\varphi$$

where $\mathop{\text{,}}(\cdot)$ denotes a macro for a comma-separated list, i.e.,

$$\mathop{\text{,}}_{i=1}^{m}(x_i)\ :=\ x_1\ \texttt{,}\ \ldots\ x_m \text{ if } m > 1 \text{ else } x_1 \text{ if } m = 1 \text{ else } \varepsilon$$

$\mu\text{Rust}_{\text{SL}}$: sequencing and binding

Sequencing. A sequence s_1 ; s_2 composes obligation contexts:

$$[\![s_1 \;;\; s_2]\!] \;=\; [\![s_2]\!] \circ [\![s_1]\!].$$

$$\frac{\Gamma, \Delta \vdash s_1 : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_1 \qquad \Gamma, \Delta \vdash s_2 : \mathtt{unit} \mid \Phi_1 \Rightarrow \Phi_2}{\Gamma, \Delta \vdash s_1 \;;\; s_2 : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_2} \text{ (SEQ)}$$

Let-binding. A local binding introduces a fresh obligation that must be subsequently discharged:

$$\Gamma, \Delta \vdash \mathtt{let}\ x = e \quad \text{acts as} \quad \Phi_0 \xrightarrow{e} \Phi_1 \xrightarrow{\cup\{x\}} \Phi_2$$

$$\frac{\Gamma, \Delta \vdash e : \tau \mid \Phi_0 \Rightarrow \Phi_1 \qquad \Gamma, \Delta \vdash x : \tau \mid \Phi_1 \cup \{x\} \Rightarrow \Phi_2}{\Gamma, \Delta \vdash \mathtt{let}\ x = e : \mathtt{unit} \mid \Phi_0 \Rightarrow \Phi_2} \text{ (LET)}$$

μRust$_{\text{SL}}$ embedding: sequencing and binding

Sequencing. Recall the (SEQ) rule, which $\mathtt{BDY}_\varphi$ will mirror:

$$\Big(\mathtt{BDY}[\tau, \Phi_0 \Rightarrow \Phi_2] \;\rightarrow\; \mathtt{STM}[\mathtt{unit}, \Phi_0 \Rightarrow \Phi_1] \; \mathtt{;} \; \mathtt{BDY}[\tau, \Phi_1 \Rightarrow \Phi_2]\Big) \in P_\Gamma,$$

$$\Big(\mathtt{BDY}[\tau, \Phi \Rightarrow \varnothing] \;\rightarrow\; \mathtt{INV}[\tau, \Phi \Rightarrow \varnothing]\Big) \in P_\Gamma$$

for all possible obligation states Φ_0, Φ_1, Φ_2, s.t. $\Phi_2 \subseteq \Phi_1 \subseteq \Phi_0$.

Let-binding. $\mathtt{STM}_\varphi$ generates a set of $\mathtt{STM}$ productions. Whenever,

$$\Gamma, \Delta \vdash e : \tau \;\mid\; \Phi_0 \Rightarrow \Phi_1 \quad \text{and} \quad \Gamma, \Delta \vdash x : \tau \;\mid\; \Phi_1 \cup \{x\} \Rightarrow \Phi_2,$$

we will add the corresponding production:

$$\Big(\mathtt{STM}[\mathtt{unit}, \Phi_0 \Rightarrow \Phi_2] \;\rightarrow\; \mathtt{let}\; x \;\mathtt{=}\; \mathtt{INV}[\tau, \Phi_0 \Rightarrow \Phi_1]\Big) \in P_\Gamma,$$

These rules ensure every word $\sigma \in \mathcal{L}(\mathtt{BDY}[\tau, \Phi \Rightarrow \varnothing])$ corresponds to a well-typed relevant μRust$_{\text{SL}}$ fragment.

Future work

- Formalize edit calculus using DiLL (*Ehrhard & Regnier*, 2003)
- Understand the connection to CMTT (*Nanevski et. al.*, 2007)
- Incrementalization and coalgebraic language intersection
- More compact embeddings and asymptotic complexity
- Lazily materialize CFG during intersection or sampling
- Extend to richer type systems, e.g., polymorphism, higher-order functions, subtyping, nested scope
- “A Tree Sampler for Bounded CFLs” (Considine, 2024) describes a uniform sampler for finite CFL intersections
- “A Word Sampler for Well-Typed Functions” (Considine, 2025) describes an embedding for simply-typed first-order functions
- Applications to proof search and property-based testing
- Try it yourself at: https://tidyparse.github.io

Acknowledgements

Thank you to the following people for inspiration!

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

Enfin, merci à Antoine Gaulin pour l'organisation !