Syntax Repair as Language Intersection

ANONYMOUS AUTHOR(S)

We introduce a new technique for correcting syntax errors in arbitrary context-free languages. Our work addresses the problem of syntax error correction, which we solve by defining a finite language that provably generates every repair within a certain edit distance. To do this, we adapt the Bar-Hillel construction from formal languages, guaranteeing this language is sound and complete with respect to a programming langauge's grammar. This technique also admits a polylogarithmic time algorithm for deciding intersection nonemptiness between CFLs and acyclic NFAs, the first of its kind in the parsing literature.

1 INTRODUCTION

During programming, one invariably encounters a recurring scenario in which the editor occupies an unparseable state, either due to an unfinished or malformed piece of code. Faced with this predicament, programmers must spend time to locate and repair the error before proceeding. We attempt to solve this problem automatically by generating a list of candidate repairs which contains with high probability the true repair, assuming this repair differs by no more than a few edits.

2 BACKGROUND

Recall that a CFG, $\mathcal{G} = \langle \Sigma, V, P, S \rangle$, is a quadruple consisting of terminals (Σ), nonterminals (V), productions ($P \colon V \longrightarrow (V \mid \Sigma)^*$), and a start symbol, ($S$). Every CFG is reducible to so-called *Chomsky Normal Form* [16], $P' \colon V \longrightarrow (V^2 \mid \Sigma)$, where every production is either (1) a binary production $w \longrightarrow xz$, or (2) a unit production $w \longrightarrow t$, where $w, x, z : V$ and $t : \Sigma$. For example:

$$G = \{\, S \to S\,S \mid (\,S\,) \mid (\,) \,\} \Longrightarrow G' = \{\, S \to Q\,R \mid S\,S \mid L\,R, \quad R \to), \quad L \to (, \quad Q \to L\,S \,\}$$

Likewise, a finite state automaton (FSA) is a quintuple $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition function, and $I, F \subseteq Q$ are the set of initial and final states, respectively. We will adhere to this notation in the following sections.

There is an equivalent characterization of the regular languages using an inductively defined datatype which is often more elegant to work with. Consider the generalized regular expression (GRE) fragment containing concatenation, conjunction and disjunction:

Definition 2.1 (Generalized Regex). Let e be an expression defined by the grammar:

$$e ::= \varnothing \mid \varepsilon \mid \Sigma \mid e \cdot e \mid e \vee e \mid e \wedge e$$

Semantically, we can interpret these expressions as denoting regular languages:

$$\begin{aligned} \mathcal{L}(\ \varnothing\ ) &= \varnothing & \qquad \mathcal{L}(\ x \cdot z\ ) &= \mathcal{L}(x) \times \mathcal{L}(z)^{1} \\ \mathcal{L}(\ \varepsilon\ ) &= \{\varepsilon\} & \mathcal{L}(\ x \vee z\ ) &= \mathcal{L}(x) \cup \mathcal{L}(z) \\ \mathcal{L}(\ a\ ) &= \{a\} & \mathcal{L}(\ x \wedge z\ ) &= \mathcal{L}(x) \cap \mathcal{L}(z) \end{aligned}$$

Brzozowski introduces the concept of differentiation, which allows us to quotient a regular language by some given prefix.

Definition 2.2 (Brzozowski, 1964). To compute the quotient $\partial_a(L) = \{b \mid ab \in L\}$, we:

$$\begin{aligned} \partial_a(\ \varnothing\ ) &= \varnothing & \qquad \delta(\ \varnothing\ ) &= \varnothing \\ \partial_a(\ \varepsilon\ ) &= \varnothing & \delta(\ \varepsilon\ ) &= \varepsilon \\ \partial_a(\ b\ ) &= \begin{cases} \varepsilon & \text{if } a = b \\ \varnothing & \text{if } a \neq b \end{cases} & \delta(\ a\ ) &= \varnothing \\ \partial_a(\ x \cdot z\ ) &= (\partial_a x) \cdot z \vee \delta(x) \cdot \partial_a z & \delta(\ x \cdot z\ ) &= \delta(x) \wedge \delta(z) \\ \partial_a(\ x \vee z\ ) &= \partial_a x \vee \partial_a z & \delta(\ x \vee z\ ) &= \delta(x) \vee \delta(z) \\ \partial_a(\ x \wedge z\ ) &= \partial_a x \wedge \partial_a z & \delta(\ x \wedge z\ ) &= \delta(x) \wedge \delta(z) \end{aligned}$$

Primarily, this gadget was designed to handle membership queries, for which purpose it has received considerable attention in recent years:

Theorem 2.3 (Recognition). *For any regex e and $\sigma : \Sigma^*$, $\sigma \in \mathcal{L}(e) \Longleftrightarrow \varepsilon \in \mathcal{L}(\partial_\sigma e)$, where:*

$$\partial_\sigma(e) : E \to E = \begin{cases} e & \text{if } \sigma = \varepsilon \\ \partial_b(\partial_a e) & \text{if } \sigma = a \cdot b, a \in \Sigma, b \in \Sigma^* \end{cases}$$

[1] Or $\{a \cdot b \mid a \in \mathcal{L}(x) \wedge b \in \mathcal{L}(z)\}$ to be more precise, however we make no distinction.

It can also be used, however, to decode witnesses. We will first focus on the nonempty disjunctive fragment, and define this process in two steps:

Theorem 2.4 (Generation). *For any nonempty $(\varepsilon, \wedge)$-free regex, e, to witness $\sigma \in \mathcal{L}(e)$:*

$$\mathtt{follow}(e) : E \to 2^{\Sigma} = \begin{cases} \{e\} & \text{if } e \in \Sigma \\ \mathtt{follow}(x) & \text{if } e = x \cdot z \\ \mathtt{follow}(x) \cup \mathtt{follow}(z) & \text{if } e = x \vee z \end{cases}$$

$$\mathtt{choose}(e) : E \to \Sigma^{+} = \begin{cases} e & \text{if } e \in \Sigma \\ \left(s \xleftarrow{\$} \mathtt{follow}(e)\right) \cdot \mathtt{choose}(\partial_s e) & \text{if } e = x \cdot z \\ \mathtt{choose}(e' \xleftarrow{\$} \{x, z\}) & \text{if } e = x \vee z \end{cases}$$

Here, we use the $\xleftarrow{\$}$ operator to denote probabilistic choice, however any deterministic choice function will also suffice to generate a witness. Now we are equipped to handle conjunction.

2.1 Language intersection

Theorem 2.5 (Bar-Hillel, 1961). *For any context-free grammar (CFG), $G = \langle V, \Sigma, P, S\rangle$, and nondeterministic finite automata, $A = \langle Q, \Sigma, \delta, I, F\rangle$, there exists a CFG $G_\cap = \langle V_\cap, \Sigma_\cap, P_\cap, S_\cap\rangle$ such that $\mathcal{L}(G_\cap) = \mathcal{L}(G) \cap \mathcal{L}(A)$.*

Definition 2.6 (Salomaa, 1973). One could construct $G_\cap$ like so,

$$\frac{q \in I \quad r \in F}{(S \to qSr) \in P_\cap}\ \surd \qquad \frac{(w \to a) \in P \qquad (q \xrightarrow{a} r) \in \delta}{(qwr \to a) \in P_\cap}\ \uparrow \qquad \frac{(w \to xz) \in P \qquad p, q, r \in Q}{(pwr \to (pxq)(qzr)) \in P_\cap}\ \bowtie$$

however most synthetic productions in $P_\cap$ will be non-generating or unreachable. This naïve method will construct a synthetic production for state pairs which are not even connected by any path, which is clearly excessive. We will instead proceed by considering a simpler problem, then construct a parse chart which efficiently computes the intersection.

3 METHOD

Our method is to treat finite language intersections as matrix exponentiation.

Theorem 3.1. *For every CFG, G, and every acyclic NFA (ANFA), A, there exists a decision procedure* $\varphi : CFG \to ANFA \to \mathbb{B}$ *such that* $\varphi(G, A) \models [\mathcal{L}(G) \cap \mathcal{L}(A) \neq \varnothing]$ *which requires* $\mathcal{O}\big((\log |Q|)^c\big)$ *time using* $\mathcal{O}\big((|V||Q|)^k\big)$ *parallel processors for some* $c, k < \infty$.

Proof. WTS there exists a path $p \rightsquigarrow r$ in A such that $p \in I, r \in F$ where $p \rightsquigarrow r \vdash S$.

There are two cases, at least one of which must hold for $w \in V$ to parse a given $p \rightsquigarrow r$ pair:

(1) p steps directly to r in which case it suffices to check $\exists a.\big((p \xrightarrow{a} r) \in \delta \wedge (w \to a) \in P\big)$, or,

(2) there is some midpoint $q \in Q$, $p \rightsquigarrow q \rightsquigarrow r$ such that $\exists x, z.\big((w \to xz) \in P \wedge \overbrace{\underbrace{p \rightsquigarrow q}_{x}, \underbrace{q \rightsquigarrow r}_{z}}^{w}\big)$.

This decomposition immediately suggests a dynamic programming solution. Let M be a matrix of type $E^{|Q|\times|Q|\times|V|}$ indexed by Q. Since we assumed δ is acyclic, there exists a topological sort of δ imposing a total order on Q such that M is strictly upper triangular (SUT). Initiate it thusly:

$$M_0[r, c, w] = \bigvee_{a \in \Sigma} \{a \mid (w \to a) \in P \wedge (q_r \xrightarrow{a} q_c) \in \delta\} \tag{1}$$

The algebraic operations $\oplus, \otimes : E^{2|V|} \to E^{|V|}$ will be defined elementwise:

$$[\ell \oplus r]_w = [\ell_w \vee r_w] \tag{2}$$

$$[\ell \otimes r]_w = \bigvee_{x,z \in V} \{\ell_x \cdot r_z \mid (w \to xz) \in P\} \tag{3}$$

By slight abuse of notation[2], we will redefine the matrix exponential over this domain as:

$$\exp(M) = \sum_{i=0}^{\infty} M_0^i = \sum_{i=0}^{|Q|} M_0^i \text{ (since } M \text{ is SUT.)} \tag{4}$$

To solve for the fixpoint, we can instead use exponentiation by squaring:

$$S(2n) = \begin{cases} M_0, & \text{if } n = 1, \\ S(n) + S(n)^2 & \text{otherwise.} \end{cases} \tag{5}$$

Therefor, we only need a maximum of $\lceil \log_2 |Q| \rceil$ sequential steps to reach the fixpoint. Finally,

$$S_\cap = \bigvee_{q \in I,\, q' \in F} \exp(M)[q, q', S] \text{ and } \varphi = [S_\cap \neq \varnothing] \tag{6}$$

To decode a witness in case of non-emptiness, we simply `choose`$(S_\cap)$. □

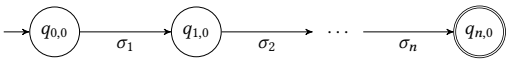
[2]Customarily, there is a $\frac{1}{k!}$ factor to suppress exploding entries, however this domain has no multiplicative inverse.

4 EXAMPLES

In this section, we will consider three examples of intersections with finite languages. First, parsing can be viewed as a special case of language intersection with an automaton accepting a single word. Second, completion can be seen as a case of intersection with terminal wildcards in known locations. Thirdly, we will consider syntax repair, in which case we will intersect a langauge representing all possible word edits to determine the location(s) and fill them with appropriate terminals.

4.1 Recognition as intersection

In the case of ordinary CFL recognition, the automaton accepts just a single word:



Given a CFG, $G' : \mathcal{G}$ in Chomsky Normal Form (CNF), we can construct a recognizer $R : \mathcal{G} \to \Sigma^n \to \mathbb{B}$ for strings $\sigma : \Sigma^n$ as follows. Let 2^V be our domain, 0 be $\varnothing$, $\oplus$ be $\cup$, and $\otimes$ be defined as:

$$X \otimes Z = \left\{ w \mid \langle x, z \rangle \in X \times Z, (w \to xz) \in P \right\} \tag{7}$$

If we define $\hat{\sigma}_r = \{w \mid (w \to \sigma_r) \in P\}$, then construct a matrix with nonterminals on the superdiagonal representing each token, $M_0[r+1 = c](G', \sigma) = \hat{\sigma}_r$, the fixpoint $M_{i+1} = M_i + M_i^2$ is uniquely determined by the superdiagonal entries. The fixedpoint iteration proceeds as follows:

$$M_0 = \begin{pmatrix} \varnothing & \hat{\sigma}_1 & \varnothing & \cdots & \varnothing \\ & \ddots & \ddots & & \\ & & & & \varnothing \\ & & & & \hat{\sigma}_n \\ \varnothing & & & & \varnothing \end{pmatrix} \Rightarrow \begin{pmatrix} \varnothing & \hat{\sigma}_1 & \Lambda & \cdots & \varnothing \\ & \ddots & \ddots & & \\ & & & & \Lambda \\ & & & & \hat{\sigma}_n \\ \varnothing & & & & \varnothing \end{pmatrix} \Rightarrow \ldots \Rightarrow M_\infty = \begin{pmatrix} \varnothing & \hat{\sigma}_1 & \Lambda & \cdots & \Lambda_\sigma^* \\ & \ddots & \ddots & & \\ & & & & \Lambda \\ & & & & \hat{\sigma}_n \\ \varnothing & & & & \varnothing \end{pmatrix}$$

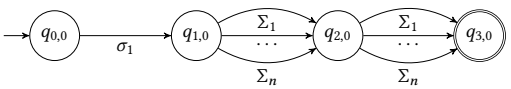
Once obtained, the proposition $[S \in \Lambda_\sigma^*]$ decides language membership, i.e., $[\sigma \in \mathcal{L}(G)]$ [3]. So far, this procedure is essentially the textbook CYK algorithm in a linear algebraic notation [26].

4.2 Completion as intersection

We can also consider a more general automaton for completing a string with holes, representing edits in fixed locations which can be filled by any terminal, which we call *completion*. In this case, the fixpoint is characterized by a system of language equations, whose solutions are the set of all sentences consistent with the template.

Definition 4.1 (Porous completion). Let $\underline{\Sigma} = \Sigma \cup \{\_\}$, where _ denotes a hole. We denote $\sqsubseteq : \Sigma^n \times \underline{\Sigma}^n$ as the relation $\{\langle \sigma', \sigma \rangle \mid \sigma_i \in \Sigma \implies \sigma'_i = \sigma_i\}$ and the set of all inhabitants $\{\sigma' : \Sigma^+ \mid \sigma' \sqsubseteq \sigma\}$ as $\mathrm{H}(\sigma)$. Given a *porous string*, $\sigma : \underline{\Sigma}^*$ we seek all syntactically valid inhabitants, i.e., $A(\sigma) = \mathrm{H}(\sigma) \cap \ell$.

Here, the FSA takes a similar shape but can have multiple arcs between subsequent states, e.g.:



This corresponds to a template with two holes, $\sigma = 1 \_ \_$. Suppose the context-free grammar is $G = \{S \to NON, O \to + \mid \times, N \to 0 \mid 1\}$. This grammar will first be rewritten into CNF as

[3] Hereinafter, we use Iverson brackets to denote the indicator function of a predicate with free variables, i.e., $[P] \Leftrightarrow \mathbb{1}(P)$.

$G' = \{S \to NL, N \to 0 \mid 1, O \to \times \mid +, L \to ON\}$. Using the powerset algebra we just defined, the matrix fixpoint $M' = M + M^2$ can be computed as follows, shown in the leftmost column below:

	2^V	$\mathbb{Z}_2^{\lvert V\rvert}$	$\mathbb{Z}_2^{\lvert V\rvert} \to \mathbb{Z}_2^{\lvert V\rvert}$
M_0	$\begin{pmatrix} \{N\} & & \\ & \{N,O\} & \\ & & \{N,O\} \end{pmatrix}$	$\begin{pmatrix} \square\blacksquare\square\square & & \\ & \square\blacksquare\blacksquare\square & \\ & & \square\blacksquare\blacksquare\square \end{pmatrix}$	$\begin{pmatrix} V_{0,1} & & \\ & V_{1,2} & \\ & & V_{2,3} \end{pmatrix}$
M_1	$\begin{pmatrix} \{N\} & \varnothing & \\ & \{N,O\} & \{L\} \\ & & \{N,O\} \end{pmatrix}$	$\begin{pmatrix} \square\blacksquare\square\square & \square\square\square\square & \\ & \square\blacksquare\blacksquare\square & \blacksquare\square\square\square \\ & & \square\blacksquare\blacksquare\square \end{pmatrix}$	$\begin{pmatrix} V_{0,1} & V_{0,2} & \\ & V_{1,2} & V_{1,3} \\ & & V_{2,3} \end{pmatrix}$
$M_2 = M_\infty$	$\begin{pmatrix} \{N\} & \varnothing & \{S\} \\ & \{N,O\} & \{L\} \\ & & \{N,O\} \end{pmatrix}$	$\begin{pmatrix} \square\blacksquare\square\square & \square\square\square\square & \square\square\square\blacksquare \\ & \square\blacksquare\blacksquare\square & \blacksquare\square\square\square \\ & & \square\blacksquare\blacksquare\square \end{pmatrix}$	$\begin{pmatrix} V_{0,1} & V_{0,2} & V_{0,3} \\ & V_{1,2} & V_{1,3} \\ & & V_{2,3} \end{pmatrix}$

The same procedure can be translated, without loss of generality, into the bit domain $(\mathbb{Z}_2^{|V|})$ using a lexicographic nonterminal ordering, however M_∞ in both 2^V and $\mathbb{Z}_2^{|V|}$ represents a decision procedure, i.e., $[S \in V_{0,3}] \Leftrightarrow [V_{0,3,3} = \blacksquare] \Leftrightarrow [A(\sigma) \neq \varnothing]$. Since $V_{0,3} = \{S\}$, we know there exists at least one solution $\sigma' \in A(\sigma)$, but M_∞ does not explicitly reveal its identity.

To extract the inhabitants, we can translate the bitwise procedure into an equation with free variables. Here, we can encode the idempotency constraint directly as $M = M^2$. We first define $X \boxtimes Z = [X_2 \wedge Z_1, \bot, \bot, X_1 \wedge Z_0]$ and $X \boxplus Z = [X_i \vee Z_i]_{i \in [0,|V|)}$, mirroring $\oplus, \otimes$ from the powerset domain, now over bitvectors. Since the unit nonterminals O, N can only occur on the superdiagonal, they may be safely ignored by $\boxtimes$. To solve for M_∞, we proceed by first computing $V_{0,2}, V_{1,3}$:

$$\begin{aligned} V_{0,2} &= V_{0,j} \cdot V_{j,2} = V_{0,1} \boxtimes V_{1,2} \\ &= [L \in V_{0,2}, \bot, \bot, S \in V_{0,2}] \\ &= [O \in V_{0,1} \wedge N \in V_{1,2}, \bot, \bot, N \in V_{0,1} \wedge L \in V_{1,2}] \\ &= [V_{0,1,2} \wedge V_{1,2,1}, \bot, \bot, V_{0,1,1} \wedge V_{1,2,0}] \end{aligned}$$

$$\begin{aligned} V_{1,3} &= V_{1,j} \cdot V_{j,3} = V_{1,2} \boxtimes V_{2,3} \\ &= [L \in V_{1,3}, \bot, \bot, S \in V_{1,3}] \\ &= [O \in V_{1,2} \wedge N \in V_{2,3}, \bot, \bot, N \in V_{1,2} \wedge L \in V_{2,3}] \\ &= [V_{1,2,2} \wedge V_{2,3,1}, \bot, \bot, V_{1,2,1} \wedge V_{2,3,0}] \end{aligned}$$

Now we solve for the corner entry $V_{0,3}$ by dotting the first row and last column, which yields:

$$\begin{aligned} V_{0,3} &= V_{0,j} \cdot V_{j,3} = (V_{0,1} \boxtimes V_{1,3}) \boxplus (V_{0,2} \boxtimes V_{2,3}) \\ &= [V_{0,1,2} \wedge V_{1,3,1} \vee V_{0,2,2} \wedge V_{2,3,1}, \bot, \bot, V_{0,1,1} \wedge V_{1,3,0} \vee V_{0,2,1} \wedge V_{2,3,0}] \end{aligned}$$

Since we only care about $V_{0,3,3} \Leftrightarrow [S \in V_{0,3}]$, we can ignore the first three entries and solve for:

$$\begin{aligned} V_{0,3,3} &= V_{0,1,1} \wedge V_{1,3,0} \vee V_{0,2,1} \wedge V_{2,3,0} \\ &= V_{0,1,1} \wedge (V_{1,2,2} \wedge V_{2,3,1}) \vee V_{0,2,1} \wedge \bot \\ &= V_{0,1,1} \wedge V_{1,2,2} \wedge V_{2,3,1} \\ &= [N \in V_{0,1}] \wedge [O \in V_{1,2}] \wedge [N \in V_{2,3}] \end{aligned}$$

Now we know that $\sigma = 1\ \underline{\mathrm{O}}\ \underline{\mathrm{N}}$ is a valid solution, and we can take the product $\{1\} \times \hat{\sigma}_2^{-1}(O) \times \hat{\sigma}_3^{-1}(N)$ to recover the inhabitants, yielding $A = \{1+0, 1+1, 1\times 0, 1\times 1\}$. In this case, since G is unambiguous, there is only one parse tree satisfying $V_{0,|\sigma|,3}$.

4.3 Repair as intersection

Finally, we are ready to consider the general case of syntax repair, in which case the edit locations are not localized but can occur anywhere in the string. In this case, we construct a lattice of all possible edit paths up to a fixed distance. This structure is called a Levenshtein automaton.

As the original construction defined by Schultz and Mihov [45] contains cycles and ε-transitions, we propose a variant which is ε-free and acyclic. Furthermore, we adopt a nominal form which supports infinite alphabets and considerably simplifies the language intersection to follow. Illustrated in Fig. 1 is an example of a small Levenshtein automaton recognizing $L(\sigma : \Sigma^5, 3)$. Unlabeled arcs accept any terminal from the alphabet, Σ. Equivalently, this transition system can be viewed as a kind of proof system within an unlabeled lattice. The following construction is equivalent to Schultz and Mihov's original Levenshtein automaton, but is more amenable to our purposes as it does not any contain ε-arcs, and instead uses skip connections to recognize consecutive deletions of varying lengths.

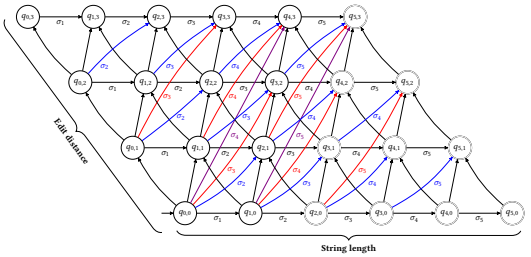


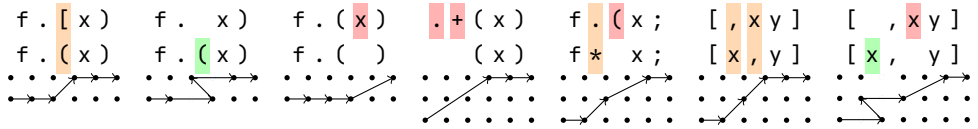
Fig. 1. NFA recognizing Levenshtein $L(\sigma : \Sigma^5, 3)$.

$$\frac{s \in \Sigma \quad i \in [0, n] \quad j \in [1, d_{\max}]}{(q_{i,j-1} \xrightarrow{s} q_{i,j}) \in \delta} \qquad \frac{s \in \Sigma \quad i \in [1, n] \quad j \in [1, d_{\max}]}{(q_{i-1,j-1} \xrightarrow{s} q_{i,j}) \in \delta}$$

$$\frac{i \in [1, n] \quad j \in [0, d_{\max}]}{(q_{i-1,j} \xrightarrow{\sigma_i} q_{i,j}) \in \delta} \qquad \frac{d \in [1, d_{\max}] \quad i \in [d+1, n] \quad j \in [d, d_{\max}]}{(q_{i-d-1,j-d} \xrightarrow{\sigma_i} q_{i,j}) \in \delta}$$

$$\frac{}{q_{0,0} \in I}\ \textsc{Init} \qquad \frac{q_{i,j} \in Q \qquad |n - i + j| \leq d_{\max}}{q_{i,j} \in F}\ \textsc{Done}$$

Each arc plays a specific role. ↖ handles insertions, ↗ handles substitutions and ⋰ handles deletions of one or more terminals. Let us consider some illustrative cases.



Note that the same patch can have multiple Levenshtein alignments. Done constructs the final states, which are all states accepting strings σ' whose Levenshtein distance $\Delta(\sigma, \sigma') \leq d_{\max}$.

To avoid creating a parallel bundle of arcs for each insertion and substitution point, we instead decorate each arc with a nominal predicate, accepting or rejecting σ_i. To distinguish this nominal variant from the original construction, we highlight the modified rules in orange below.

$$\frac{i \in [0, n] \quad j \in [1, d_{\max}]}{(q_{i,j-1} \xrightarrow{[\neq \sigma_{i+1}]} q_{i,j}) \in \delta} \qquad \frac{i \in [1, n] \quad j \in [1, d_{\max}]}{(q_{i-1,j-1} \xrightarrow{[\neq \sigma_i]} q_{i,j}) \in \delta}$$

$$\frac{i \in [1, n] \quad j \in [0, d_{\max}]}{(q_{i-1,j} \xrightarrow{[= \sigma_i]} q_{i,j}) \in \delta} \qquad \frac{d \in [1, d_{\max}] \quad i \in [d+1, n] \quad j \in [d, d_{\max}]}{(q_{i-d-1,j-d} \xrightarrow{[= \sigma_i]} q_{i,j}) \in \delta}$$

Nominalizing the NFA eliminates the creation of $e = 2(|\Sigma| - 1) \cdot |\sigma| \cdot d_{\max}$ unnecessary arcs over the entire Levenshtein automaton and drastically reduces the representation size, but does not affect the underlying semantics. Thus, it is important to first nominalize the automaton before proceeding to avoid a large blowup in the intermediate grammar.

Now, we need to order the automata states by increasing shortest-path distance from q_0. One approach would be to topologically sort the adjacency matrix using a max-plus semiring. While some form of sorting is unavoidable for arbitrary NFAs, if we assume our structure is a Levenshtein automaton, we can simply enumerate the state space by increasing Manhattan distance from the origin using, e.g., the Cantor pairing function to construct a valid ordering.

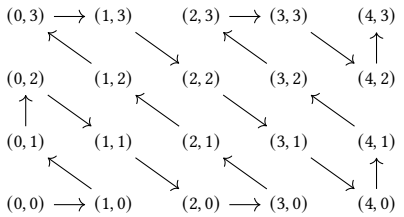


Fig. 2. Pairing function over $L(\sigma : \Sigma^4, 3)$.

5 MEASURING THE LANGUAGE INTERSECTION

We will now attempt to put a probability distribution over the language intersection. We will start with a few cursory but illumative approaches, then proceed towards a more refined solution.

5.1 Exact enumeration

A brute force solution would be to generate every path and rank every one by its probability. It should be obvious why is unviable due its worst case complexity, but bears mentioning due to its global optimality. In certain cases, it can be realized when the intersection language is small.

To enumerate, we first need $|\mathcal{L}(e)|$, which is denoted $|e|$ for brevity.

Definition 5.1 (Cardinality). $$|e| : E \to \mathbb{N} = \begin{cases} 1 & \text{if } R \in \Sigma \\ x \times z & \text{if } e = x \cdot z \\ x + z & \text{if } e = x \vee z \end{cases}$$

Theorem 5.2 (Enumeration). *To enumerate, invoke* $\bigcup_{i=0}^{|R|} \{\mathit{enum}(R, i)\}$*:*

$$\mathit{enum}(e, n) : E \times \mathbb{N} \to \Sigma^* = \begin{cases} e & \text{if } R \in \Sigma \\ \mathit{enum}\left(x, \lfloor \frac{n}{|z|} \rfloor\right) \cdot \mathit{enum}\left(z,\ n \bmod |z|\right) & \text{if } e = x \cdot z \\ \mathit{enum}\left((x, z)_{\min(1, \lfloor \frac{n}{|x|} \rfloor)}, n - |x| \min(1, \lfloor \frac{n}{|x|} \rfloor)\right) & \text{if } e = x \vee z \end{cases}$$

5.2 Mode collapse

Ordinarily, we would use top-down PCFG sampling, however in the case of non-recursive CFGs, this method is highly degenerate, exhibiting poor sample diversity. Consider an illustrative pathological case for top-down ancestral (TDA) sampling:

$$S \to A\ B\ (0.9999) \qquad S \to C\ C\ (0.0001)$$

$$A \to a\ (1) \qquad B \to b\ (1) \qquad C \to a \left(\frac{1}{26}\right) \mid \ldots \mid z \left(\frac{1}{26}\right)$$

TDA sampling will almost always generate the string *ab*, but most of the language is concealed in the hidden branch, $S \to CC$. Although contrived example, it illustrates precisely why TDA sampling is unviable: we want a sampler that matches the true distribution over the finite CFL, not the PCFG's local approximation thereof.

5.3 Ambiguity

Another approach would be to sample trees and rerank them by their PCFG score. More pernicious is the issue of ambiguity. Since the CFG can be ambiguous, this causes certain repairs to be overrepresented, resulting in a subtle bias. Consider for example,

Lemma 5.3. *If the FSA, α, is ambiguous, then the intersection grammar, $G_\cap$, can be ambiguous.*

Proof. Let ℓ be the language defined by $G = \{S \to LR, L \to (, R \to)\}$, where $\alpha = L(\underset{\sim}{\sigma}, 2)$, the broken string $\underset{\sim}{\sigma}$ is)(, and $\mathcal{L}(G_\cap) = \ell \cap \mathcal{L}(\alpha)$. Then, $\mathcal{L}(G_\cap)$ contains the following two identical repairs:)() with the parse $S \to q_{00}Lq_{21}\ q_{21}Rq_{22}$, and () with the parse $S \to q_{00}Lq_{11}\ q_{11}Rq_{22}$. □

We would like the underlying sample space to be a proper set, *not* a multiset.

REFERENCES

[1] Michael D Adams, Celeste Hollenbeck, and Matthew Might. 2016. On the complexity and performance of parsing with derivatives. In Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. 224–236.

[2] Alfred V Aho and Thomas G Peterson. 1972. A minimum distance error-correcting parser for context-free languages. SIAM J. Comput. 1, 4 (1972), 305–312.

[3] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. 2021. Self-supervised bug detection and repair. Advances in Neural Information Processing Systems 34 (2021), 27865–27876. https://arxiv.org/pdf/2105.12787.pdf

[4] Valentin Antimirov. 1996. Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science 155, 2 (1996), 291–319.

[5] Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. 1961. On formal properties of simple phrase structure grammars. Sprachtypologie und Universalienforschung 14 (1961), 143–172.

[6] Daniella Bar-Lev, Tuvi Etzion, and Eitan Yaakobi. 2021. On Levenshtein Balls with Radius One. In 2021 IEEE International Symposium on Information Theory (ISIT). 1979–1984. https://doi.org/10.1109/ISIT45174.2021.9517922

[7] Leonor Becerra-Bonache, Colin de La Higuera, Jean-Christophe Janodet, and Frédéric Tantini. 2008. Learning Balls of Strings from Edit Corrections. Journal of Machine Learning Research 9, 8 (2008).

[8] Richard Beigel and William Gasarch. [n.d.]. A Proof that if $L = L_1 \cap L_2$ where L_1 is CFL and L_2 is Regular then L is Context Free Which Does Not use PDA's. http://www.cs.umd.edu/~gasarch/BLOGPAPERS/cfg.pdf

[9] Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. 2014. Automata theory in nominal sets. Logical Methods in Computer Science 10 (2014).

[10] Janusz A Brzozowski. 1962. Canonical regular expressions and minimal state graphs for definite events. In Proc. Symposium of Mathematical Theory of Automata. 529–561.

[11] Janusz A Brzozowski. 1964. Derivatives of regular expressions. Journal of the ACM (JACM) 11, 4 (1964), 481–494.

[12] Janusz A. Brzozowski and Ernst Leiss. 1980. On equations for regular languages, finite automata, and sequential networks. Theoretical Computer Science 10, 1 (1980), 19–35.

[13] David Chiang. 2007. Hierarchical phrase-based translation. computational linguistics 33, 2 (2007), 201–228.

[14] David Chiang, Peter Cholak, and Anand Pillay. 2023. Tighter bounds on the expressivity of transformer encoders. In International Conference on Machine Learning. PMLR, 5544–5562. https://proceedings.mlr.press/v202/chiang23a/chiang23a.pdf

[15] Nadezhda Chirkova and Sergey Troshin. 2021. Empirical study of transformers for source code. In Proceedings of the 29th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering. 703–715.

[16] Noam Chomsky. 1959. On certain formal properties of grammars. Information and control 2, 2 (1959), 137–167.

[17] Loris D'Antoni and Margus Veanes. 2014. Minimization of symbolic automata. In Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 541–553.

[18] Dingding Dong, Nitya Mani, and Yufei Zhao. 2023. On the number of error correcting codes. Combinatorics, Probability and Computing (2023), 1–14. https://doi.org/10.1017/S0963548323000111

[19] Dawn Drain, Chen Wu, Alexey Svyatkovskiy, and Neel Sundaresan. 2021. Generating bug-fixes using pretrained transformers. In Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming. 1–8.

[20] Philippe Duchon, Philippe Flajolet, et al. 2004. Boltzmann samplers for the random generation of combinatorial structures. Combinatorics, Probability and Computing 13, 4-5 (2004), 577–625.

[21] Jay Earley. 1970. An efficient context-free parsing algorithm. Commun. ACM 13, 2 (1970), 94–102.

[22] David Eppstein. 2014. k-best enumeration. arXiv preprint arXiv:1412.5075 (2014).

[23] Denis Firsov and Tarmo Uustalu. 2015. Certified normalization of context-free grammars. In Proceedings of the 2015 Conference on Certified Programs and Proofs. 167–174.

[24] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. 1992. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. Discrete Applied Mathematics 39, 3 (1992), 207–229.

[25] Seymour Ginsburg and H Gordon Rice. 1962. Two families of languages related to ALGOL. Journal of the ACM (JACM) 9, 3 (1962), 350–371.

[26] Joshua Goodman. 1999. Semiring parsing. Computational Linguistics 25, 4 (1999), 573–606. https://aclanthology.org/J99-4004.pdf

[27] Vivek Gore, Mark Jerrum, Sampath Kannan, Z Sweedyk, and Steve Mahaney. 1997. A quasi-polynomial-time algorithm for sampling words from a context-free language. Information and Computation 134, 1 (1997), 59–74.

[28] Matthew Hague, Artur Jeż, and Anthony W Lin. 2024. Parikh's Theorem Made Symbolic. Proceedings of the ACM on Programming Languages 8, POPL (2024), 1945–1977.

[29] Timothy Hickey and Jacques Cohen. 1983. Uniform random generation of strings in a context-free language. SIAM J. Comput. 12, 4 (1983), 645–655.

[30] Liang Huang and David Chiang. 2005. Better k-best parsing. In Proceedings of the Ninth International Workshop on Parsing Technology. 53–64.

[31] Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In Proceedings of the 45th annual meeting of the association of computational linguistics. 144–151.

[32] E. T. Irons. 1963. An Error-Correcting Parse Algorithm. Commun. ACM 6, 11 (nov 1963), 669–673. https://doi.org/10.1145/368310.368385

[33] Adam Kiezun, Vijay Ganesh, Philip J Guo, Pieter Hooimeijer, and Michael D Ernst. 2009. HAMPI: a solver for string constraints. In Proceedings of the eighteenth international symposium on Software testing and analysis. 105–116.

[34] Lillian Lee. 2002. Fast context-free grammar parsing requires fast boolean matrix multiplication. Journal of the ACM (JACM) 49, 1 (2002), 1–15. https://arxiv.org/pdf/cs/0112018.pdf

[35] Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady, Vol. 10. 707–710. https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf

[36] Alexander K Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K Mansinghka. 2023. Sequential monte carlo steering of large language models using probabilistic programs. arXiv preprint arXiv:2306.03081 (2023).

[37] William Merrill, Ashish Sabharwal, and Noah A Smith. 2022. Saturated transformers are constant-depth threshold circuits. Transactions of the Association for Computational Linguistics 10 (2022), 843–856.

[38] Matthew Might, David Darais, and Daniel Spiewak. 2011. Parsing with derivatives: a functional pearl. ACM sigplan notices 46, 9 (2011), 189–195.

[39] Mark-Jan Nederhof and Giorgio Satta. 2004. The language intersection problem for non-recursive context-free grammars. Information and Computation 192, 2 (2004), 172–184.

[40] Rohit J. Parikh. 1966. On Context-Free Languages. J. ACM 13, 4 (oct 1966), 570–581. https://doi.org/10.1145/321356.321364

[41] Terence J. Parr and Russell W. Quong. 1995. ANTLR: A predicated-LL (k) parser generator. Software: Practice and Experience 25, 7 (1995), 789–810.

[42] Clemente Pasti, Andreas Opedal, Tiago Pimentel, Tim Vieira, Jason Eisner, and Ryan Cotterell. 2023. On the Intersection of Context-Free and Regular Languages. In Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, Andreas Vlachos and Isabelle Augenstein (Eds.). Association for Computational Linguistics, Dubrovnik, Croatia, 737–749. https://doi.org/10.18653/v1/2023.eacl-main.52

[43] Itiroo Sakai. 1961. Syntax in universal translation. In Proceedings of the International Conference on Machine Translation and Applied Language Analysis.

[44] Georgios Sakkas, Madeline Endres, Philip J Guo, Westley Weimer, and Ranjit Jhala. 2022. Seq2Parse: neurosymbolic parse error repair. Proceedings of the ACM on Programming Languages 6, OOPSLA2 (2022), 1180–1206.

[45] Klaus U Schulz and Stoyan Mihov. 2002. Fast string correction with Levenshtein automata. International Journal on Document Analysis and Recognition 5 (2002), 67–85.

[46] Elizabeth Scott and Adrian Johnstone. 2010. GLL parsing. Electronic Notes in Theoretical Computer Science 253, 7 (2010), 177–189.

[47] Kensen Shi, David Bieber, and Charles Sutton. 2020. Incremental sampling without replacement for sequence models. In International Conference on Machine Learning. PMLR, 8785–8795.

[48] Michalis K Titsias and Christopher Yau. 2017. The Hamming ball sampler. J. Amer. Statist. Assoc. 112, 520 (2017), 1598–1611.

[49] Leslie G Valiant. 1975. General context-free recognition in less than cubic time. Journal of computer and system sciences 10, 2 (1975), 308–315. http://people.csail.mit.edu/virgi/6.s078/papers/valiant.pdf

[50] Alexander William Wong, Amir Salimi, Shaiful Chowdhury, and Abram Hindle. 2019. Syntax and Stack Overflow: A methodology for extracting a corpus of syntax errors and fixes. In 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 318–322.

[51] Michihiro Yasunaga and Percy Liang. 2021. Break-it-fix-it: Unsupervised learning for program repair. In International Conference on Machine Learning. PMLR, 11941–11952.

[52] Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning. 320–331.