Syntax Repair as Idempotent Tensor Completion

Breandan Considine[1], Jin Guo[1], and Xujie Si[2]

[1] McGill University, Montréal, QC H2R 2Z4, Canada
{breandan.considine@mail, jguo@cs}.mcgill.ca
[2] University of Toronto, Toronto, ON, M5S 1A1 Canada
six@utoronto.ca

Abstract. We introduce a new technique for correcting syntax errors in arbitrary context-free languages. To do so, we reduce CFL recognition onto a Boolean tensor completion and compare various techniques for introducing the holes, and solving for their inhabitants. Our technique has practical applications for real-time syntax correction in programming languages.

Keywords: Error correction · CFL reachability · Langauge games.

1 Introduction

Syntax repair is the problem of taking a grammar and a malformed string, and modifying the string so it conforms to the grammar. Prior work has been devoted to fixing syntax errors using handcrafted heuristics. We take a first-principles approach that makes no assumptions about the string or grammar and focuses on accuracy and end-to-end latency. The result is a tool that is applicable to any context-free and conjunctive language, and which is provably sound and complete up to a Levenshtein bound.

1.1 Problem

Syntax repair can be treated as a language intersection problem between a context-free language (CFL) and a regular language.

Definition 1 (Bounded Levenshtein-CFL reachability). *Given a CFL ℓ and an invalid string $\underset{\sim}{\sigma} : \ell^{\complement}$, the BCFLR problem is to find every valid string reachable within d edits of $\underset{\sim}{\sigma}$, i.e., letting Δ be the Levenshtein metric and $L(\underset{\sim}{\sigma}, d) \coloneqq \{\sigma \mid \Delta(\underset{\sim}{\sigma}, \sigma) \leq d\}$, we seek to find $L(\underset{\sim}{\sigma}, d) \cap \ell$.*

To solve this problem, we will first pose a simpler problem that only considers intersections with a finite language, then turn our attention back to BCFLR.

Definition 2 (Porous completion). *Let $\underline{\Sigma} \coloneqq \Sigma \cup \{\_\}$, where $\_$ denotes a hole. We denote $\sqsubseteq\colon \Sigma^n \times \underline{\Sigma}^n$ as the relation $\{\langle \sigma', \sigma\rangle \mid \sigma_i \in \Sigma \implies \sigma'_i = \sigma_i\}$ and the set of all inhabitants $\{\sigma' \mid \sigma' \sqsubseteq \sigma\}$ as $H(\sigma)$. Given a porous string, $\sigma : \underline{\Sigma}^*$ we seek all syntactically admissible inhabitants, i.e., $A(\sigma) \coloneqq H(\sigma) \cap \ell$.*

$A(\sigma)$ is often a large-cardinality set, so we want a procedure which returns the most likely members first, without exhaustive enumeration. More precisely,

Definition 3 (Ranked repair). *Given a finite language $\ell^\cap := L(\underset{\sim}{\sigma}, d) \cap \ell$ and a probabilistic language model $P_\theta : \Sigma^* \to [0,1] \subset \mathbb{R}$, the ranked repair problem is to find the top-k repairs by likelihood under the language model. That is,*

$$R(\ell^\cap, P_\theta) := \underset{\{\boldsymbol{\sigma} | \boldsymbol{\sigma} \subseteq \ell^\cap, |\boldsymbol{\sigma}| \leq k\}}{\operatorname{argmax}} \sum_{\sigma \in \boldsymbol{\sigma}} \prod_{i=1}^{|\sigma|} P_\theta(\sigma_i \mid \sigma_{1\ldots i})^{\frac{1}{|\sigma|}} \tag{1}$$

We want a procedure $\hat{R}$, minimizing $\mathbb{E}_{G,\sigma}\big[D_{KL}(\hat{R} \parallel R)\big]$ and wallclock runtime.

Our key innovation and the core problem this paper tackles is, given $\underset{\sim}{\sigma}, d, P_\theta$, to approximate $R(\ell^\cap, P_\theta)$ while minimizing latency and maximizing accuracy. We will first give an example, then dive into the theory.

1.2 Background

Recall that a CFG is a quadruple consisting of terminals (Σ), nonterminals (V), productions ($P \colon V \to (V \mid \Sigma)^*$), and a start symbol, ($S$). Every CFG is reducible to *Chomsky Normal Form*, $P' \colon V \to (V^2 \mid \Sigma)$, in which every P takes one of two forms, either $w \to xz$, or $w \to t$, where $w, x, z : V$ and $t : \Sigma$. For example:

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

Given a CFG, $G' : \mathbb{G} = \langle \Sigma, V, P, S \rangle$ in CNF, we can construct a recognizer $R : \mathbb{G} \to \Sigma^n \to \mathbb{B}$ for strings $\sigma : \Sigma^n$ as follows. Let 2^V be our domain, 0 be $\varnothing$, $\oplus$ be $\cup$, and $\otimes$ be defined as:

$$X \otimes Z := \{\ w \mid \langle x, z \rangle \in X \times Z, (w \to xz) \in P\ \} \tag{2}$$

If we define $\sigma_r := \{w \mid (w \to \sigma_r) \in P\}$, then construct a matrix with nonterminals on the superdiagonal representing each token, $M_{r+1=c}(G', e) := \sigma_r$ and solve for the fixpoint $M_{i+1} = M_i + M_i^2$,

$$M_0 := \begin{pmatrix} \varnothing & \sigma_1 & \varnothing & \varnothing \\ & \ddots & \ddots & \\ & & \ddots & \varnothing \\ & & & \sigma_n \\ \varnothing & \cdots & & \varnothing \end{pmatrix} \Rightarrow \begin{pmatrix} \varnothing & \sigma_1 & \Lambda & \varnothing \\ & \ddots & \ddots & \\ & & \ddots & \Lambda \\ & & & \sigma_n \\ \varnothing & \cdots & & \varnothing \end{pmatrix} \Rightarrow \ldots \Rightarrow M_\infty = \begin{pmatrix} \varnothing & \sigma_1 & \Lambda & \Lambda_\sigma^* \\ & \ddots & \ddots & \\ & & \ddots & \Lambda \\ & & & \sigma_n \\ \varnothing & \cdots & & \varnothing \end{pmatrix}$$

we obtain the recognizer, $R(G', \sigma) := [S \in \Lambda_\sigma^*] \Leftrightarrow [\sigma \in \mathcal{L}(G)]$ [3].

Since $\bigoplus_{c=1}^{n} M_{r,c} \otimes M_{c,r}$ has cardinality bounded by $|V|$, it can be represented as $\mathbb{Z}_2^{|V|}$ using the characteristic function, $\mathbb{1}$. A concrete example is shown in § 1.3.

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

1.3 Example

Let us consider an example with two holes, $\sigma = 1 \_ \_$, and the grammar being $G \coloneqq \{S \to NON, O \to + \mid \times, N \to 0 \mid 1\}$. This can be rewritten into CNF as $G' \coloneqq \{S \to NL, N \to 0 \mid 1, O \to \times \mid +, L \to ON\}$. Using the algebra where $\oplus \coloneqq \cup$, $X \otimes Z \coloneqq \left\{\ w \mid \langle x, z\rangle \in X \times Z, (w \to xz) \in P\ \right\}$, the fixpoint $M' = M + M^2$ can be computed as follows, shown in the leftmost column:

	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_∞	$\begin{pmatrix} \{N\} & \varnothing & \{S\} \\ & \{N,O\} & \{L\} \\ & & \{N,O\} \end{pmatrix}$	$\begin{pmatrix} \square\blacksquare\square\square & \square\square\square\square & \square\square\square\blacksquare \\ & \square\blacksquare\blacksquare\square & \blacksquare\square\square\square \\ & & \square\blacksquare\blacksquare\square \end{pmatrix}$	$\begin{pmatrix} V_{0,1} & V_{0,2} & V_{0,3} \\ & V_{1,2} & V_{1,3} \\ & & V_{2,3} \end{pmatrix}$

The same procedure can be translated, without loss of generality, into the bit domain ($\mathbb{Z}_2^{|V|}$) using a lexicographic ordering, however these both are recognizers. That is to say, $[S \in V_{0,3}] \Leftrightarrow [V_{0,3,3} = 1] \Leftrightarrow [A(\sigma) \neq \varnothing]$. Since $V_{0,3} = \{S\}$, we know there is at least one $\sigma' \in A(\sigma)$, but M_∞ does not reveal its identity.

In order 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 $X \boxtimes Z \coloneqq [X_2 \wedge Z_1, \bot, \bot, X_1 \wedge Z_0]$ and $X \boxplus Z \coloneqq [X_i \vee Z_i]_{i \in [0,|V|)}$. Since the unit nonterminals O, N can only occur on the superdiagonal, they may be safely ignored by $\otimes$. To solve for M_∞, we proceed by first computing $V_{0,2}, V_{0,3}$ as follows:

$$
\begin{aligned}
V_{0,2} &= V_{0,j} \cdot V_{j,2} = V_{0,1} \boxtimes V_{1,2} && (3)\\
&= [L \in V_{0,2}, \bot, \bot, S \in V_{0,2}] && (4)\\
&= [O \in V_{0,1} \wedge N \in V_{1,2}, \bot, \bot, N \in V_{0,1} \wedge L \in V_{1,2}] && (5)\\
&= [V_{0,1,2} \wedge V_{1,2,1}, \bot, \bot, V_{0,1,1} \wedge V_{1,2,0}] && (6)
\end{aligned}
$$

$$
\begin{aligned}
V_{1,3} &= V_{1,j} \cdot V_{j,3} = V_{1,2} \boxtimes V_{2,3} && (7)\\
&= [L \in V_{1,3}, \bot, \bot, S \in V_{1,3}] && (8)\\
&= [O \in V_{1,2} \wedge N \in V_{2,3}, \bot, \bot, N \in V_{1,2} \wedge L \in V_{2,3}] && (9)\\
&= [V_{1,2,2} \wedge V_{2,3,1}, \bot, \bot, V_{1,2,1} \wedge V_{2,3,0}] && (10)
\end{aligned}
$$

Now we can solve for $V_{0,3}$ by taking the bitwise dot product:

$$V_{0,3} = V_{0,j} \cdot V_{j,3} = V_{0,1} \boxtimes V_{1,3} \boxplus V_{0,2} \boxtimes V_{2,3} \tag{11}$$
$$= [V_{0,1,2} \wedge V_{1,3,1} \vee V_{0,2,2} \wedge V_{2,3,1}, \bot, \bot, V_{0,1,1} \wedge V_{1,3,0} \vee V_{0,2,1} \wedge V_{2,3,0}] \tag{12}$$

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

$$V_{0,3,3} = V_{0,1,1} \wedge V_{1,3,0} \vee V_{0,2,1} \wedge V_{2,3,0} \tag{13}$$
$$= V_{0,1,1} \wedge (V_{1,2,2} \wedge V_{2,3,1}) \vee V_{0,2,1} \wedge \bot \tag{14}$$
$$= V_{0,1,1} \wedge V_{1,2,2} \wedge V_{2,3,1} \tag{15}$$
$$= [N \in V_{0,1}] \wedge [O \in V_{1,2}] \wedge [N \in V_{2,3}] \tag{16}$$

Now we know that $\sigma = 1\ \underline{\mathrm{O}}\ \underline{\mathrm{N}}$ is a valid solution, and therefor we can take the product $\{1\} \times \sigma_r^{-1}(O) \times \sigma_r^{-1}(N)$ to recover the admissible set, yielding $A(\sigma) = \{1+0, 1+1, 1\times 0, 1 \times 1\}$. In this case, since G is unambiguous, there is only one parse tree satisfying $V_{0,|\sigma|,|\sigma|}$, but in general, there can be multiple valid parse trees, in which case we can decode them incrementally.

1.4 Semiring Algebras

There are a number of alternate semirings which solve the same problem. A first approach requires solving for $A(\sigma)$ using a semiring algebra, and propagating the values from the bottom-up as a string to a list of strings. Letting $D = V \to \mathcal{P}(\Sigma^*)$, we define $\oplus, \otimes : D \times D \to D$. Initially, we have $p(s : \Sigma) \coloneqq \{v \mid [v \to s] \in P\}$ and $p(\_) := \bigcup_{s \in \Sigma} p(s)$, then we compute the fixpoint using the following algebra:

$$X \oplus Z := \{v \to \big(X(v) \cup Z(v)\big) \mid v \in V\} \tag{17}$$

$$X \otimes Z := \bigoplus_{w,x,z} \big\{w \to (l + r) \mid [w \to xz] \in P, \langle l, r\rangle \in X(x) \times Z(x)\big\} \tag{18}$$

After the fixpoint M_∞ is attained, the solutions can be read off via $M_\infty[0, |\sigma|](S)$. The issue here is an exponential growth in cardinality when eagerly computing the Cartesian product, which becomes impractical for even small strings. We can make this encoding more compact by propagating an algebraic data type (ADT) $\mathbb{T}_2$ using the operations $\oplus, \otimes : 2^{\mathbb{T}_2} \times 2^{\mathbb{T}_2} \to 2^{\mathbb{T}_2}$ as follows:

$$X \oplus Z := \{\mathbb{T}_2(k, Q_x \cup Q_z) \mid (k, Q)_X \bowtie_k (k, Q)_Z\} \tag{19}$$

$$X \otimes Z := \bigoplus_{w,x,z} \big\{\mathbb{T}_2(w, \{\langle T_x, T_z\rangle\}) \mid [w \to xz] \in P, x \in \pi_1(X), z \in \pi_1(Z)\big\} \tag{20}$$

Decoding, then becomes a matter of enumerating binary trees from the ADT using a recursive choice function that emits a sequence of strings satisfying $A(\sigma)$, with the type signature $\mathcal{C} : \mathbb{T}_2 \to (\mathbb{N} \to \Sigma^*)$ defined as follows:

$$\mathcal{C}(t : \mathbb{T}_2) := \begin{cases} \pi_1(t) & \text{if } \pi_2(t) = \varnothing, \text{ or} \\ \{x + z \mid \langle X, Z \rangle \in \pi_2(t), x \in \mathcal{C}(X), z \in \mathcal{C}(Z)\} & \text{otherwise.} \end{cases}$$

In our experiments, we will provide a comparison of the performance of the SAT algebra and these two semirings.

1.5 Bounded CFL Reachability

Now, let us return to the problem of bounded CFL reachability. The question arises, where do we put the holes? Below is an example illustrating this procedure on a single Python snippet.

1. `d = sum([foo(i] for i in vals))`
2. `d = sum ( [ foo ( i ] for i in vals ) )`
3. `w = w ( [ w ( w ] for w in w ) )`
4. `w = w ( [ w ( w ] for w in w ) )`
5. `_ = _ ( [ w ( w ] for w in w ) _`
   `w _ w ( [ _ ( w ] for w _ w ) )`
   ...
6. `w = w ( [ w ( _ w _ for w in w _ )`
   `+ ) ]`
7. (a) `d = sum([foo(+i) for i in vals])`
8. (b) `d = sum([foo(i) for i in vals])`

The initial broken string, `d = sum([foo(i] for i in vals))` (1), is first tokenized using a lexer to obtain the sequence in (2). Lexical tokens containing identifiers are abstracted in step (3), and interleaved with the empty token in step (4). We then sample hole configurations without replacement in step (5), many of which will have no admissible solutions. Eventually, the solver will discover an admissible solution, as seen in step (6). This solution is then used to generate a patch, which is applied to the original string in step (7a), then reduced to its minimal form in step (7b), and sampling is repeated until all possibilities are exhausted or a predetermined timeout expires.

1.6 Sampling the Levenshtein ball without replacement in $\mathcal{O}(1)$

Now that we have a reliable method to synthesize admissible completions for strings containing holes, i.e., fix *localized* errors, $F : (\mathcal{G} \times \underline{\Sigma}^n) \to \{\Sigma^n\} \subseteq \mathcal{L}(\mathcal{G})$, how can we use F to repair some unparseable string, i.e., $\sigma_1 \ldots \sigma_n : \Sigma^n \cap \mathcal{L}(\mathcal{G})^{\complement}$ where the holes' locations are unknown? Three questions stand out in particular:

how many holes are needed to repair the string, where should we put those holes, and how ought we fill them to obtain a parseable $\tilde{\sigma} \in \mathcal{L}(\mathcal{G})$?

One plausible approach would be to draw samples with a PCFG, minimizing tree-edit distance, however these are computationally expensive metrics and approximations may converge poorly. A more efficient strategy is to sample string perturbations, $\boldsymbol{\sigma} \sim \Sigma^{n \pm q} \cap \Delta_q(\underset{\sim}{\sigma})$ uniformly across the Levenshtein q-ball centered on $\underset{\sim}{\sigma}$, i.e., the space of all admissible edits with Levenshtein distance $\leq q$.

To implement this strategy, we first construct a surjection $\varphi^{-1} : \mathbb{Z}_2^m \twoheadrightarrow \Delta_q(\underset{\sim}{\sigma})$ from bitvectors to Levenshtein edits over $\underset{\sim}{\sigma}, \Sigma$, sample bitvectors without replacement using a characteristic polynomial, then decode the resulting bitvectors into Levenshtein edits. This ensures the sampler eventually visits every Levenshtein edit at least exactly once and at most approximately once, without needing to store any samples, and discovers a steady stream of admissible edits throughout the solving process, independent of the grammar or string under repair.

More specifically, we employ a pair of [un]tupling functions $\kappa, \rho : \mathbb{N}^k \leftrightarrow \mathbb{N}$ which are (1) bijective (2) maximally compact (3) computationally tractable (i.e., closed form inverses). κ will be used to index $\binom{n}{k}$[2]-combinations via the Maculay representation [3] and ρ will index Σ^k tuples, but is slightly more tricky to define. To maximize compactness, there is an elegant pairing function courtesy of Szudzik [5], which enumerates concentric square shells over the plane $\mathbb{N}^2$ and can be generalized to hypercubic shells in $\mathbb{N}^k$.

Although $\langle \kappa, \rho \rangle$ could be used directly to exhaustively search the Levenshtein ball, they are temporally biased samplers due to lexicographic ordering. Rather, we would prefer a path that uniformly visits every fertile subspace of the Levenshtein ball over time regardless of the grammar or string in question: subsequences of $\langle \kappa, \rho \rangle$ should discover valid repairs with frequency roughly proportional to the filtration rate, i.e., the density of the admissible set relative to the Levenshtein ball. These additional constraints give rise to two more criteria: (4) ergodicity and (5) periodicity.

To achieve ergodicity, we permute the elements of $\binom{n}{k} \times \Sigma^k$ using a finite field with a characteristic polynomial C of degree $m \coloneqq \lceil \log_p \binom{n}{k} |\Sigma_\varepsilon|^k \rceil$. By choosing C to be some irreducible polynomial, one ensures the path has the mixing properties we desire, e.g., suppose $U : \mathbb{Z}_2^{m \times m}$ is a matrix whose structure is depicted to the right, wherein C represents a primitive polynomial over $\mathbb{Z}_2^m$ with known coefficients $C_{1 \ldots m}$ and semiring operators $\oplus \coloneqq + \pmod 2, \otimes \coloneqq \wedge, \top \coloneqq 1, \circ \coloneqq 0$. Since C is primitive, the sequence $\mathbf{R} = (U^{0 \ldots 2^m - 1} Y)$ must have *full periodicity*, i.e., for all $i, j \in [0, 2^m)$, $\mathbf{R}_i = \mathbf{R}_j \Rightarrow i = j$. To uniformly sample $\boldsymbol{\sigma}$ without replacement, we construct a partial surjective function from $\mathbb{Z}_2^m$ onto the Lev-

$$U^\intercal Y = \begin{pmatrix} C_1 & \cdots & C_m \\ \top & \circ & \circ \\ \circ & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \circ & \circ \; \top & \circ \end{pmatrix}^t \begin{pmatrix} Y_1 \\ \vdots \\ \vdots \\ Y_m \end{pmatrix}$$

[2] Following Stirling, we use $\binom{n}{d}$ to denote the set of all d-element subsets of $\{1, \ldots, n\}$.

enshtein ball, $\mathbb{Z}_2^m \rightharpoonup \left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^d$, cycle over $\mathbf{R}$, then discard samples which have no witness in $\left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^d$.

This procedure requires $\mathcal{O}(1)$ per sample and roughly $\binom{n}{d}|\Sigma_\varepsilon|^d$ samples to exhaustively search $\left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^d$. Its acceptance rate $p^{-m}\binom{n}{d}|\Sigma_\varepsilon|^d$ can be slightly improved with a more suitable base p, however this introduces some additional complexity and so we elected to defer this optimization.

In addition to its statistically desirable properties, our sampler has the practical benefit of being trivially parallelizable using the leapfrog method, i.e., given p independent processors, each one can independently check $[\varphi^{-1}(\langle \kappa, \rho \rangle^{-1}(\mathbf{R}_i), \underset{\sim}{\sigma}) \in \mathcal{L}(\mathcal{G})]$ where $i \equiv p_j \pmod p$. This procedure linearly scales with the number of processors, exhaustively searching $\Delta_q(\underset{\sim}{\sigma})$ in p^{-1} of the time required by a single processor, or alternately drawing p times as many samples in the same amount of time.

To admit variable-length edits and enable deletion, we first define a ε^+-production and introduce it to the right- and left-hand side of each terminal in a unit production in our grammar, $\mathcal{G}$:

$$\frac{\mathcal{G} \vdash \varepsilon \in \Sigma}{\mathcal{G} \vdash (\varepsilon^+ \to \varepsilon \mid \varepsilon\, \varepsilon^+) \in P}\ \varepsilon\text{-DUP} \qquad \frac{\mathcal{G} \vdash (A \to B) \in P}{\mathcal{G} \vdash (A \to B\, \varepsilon^+ \mid \varepsilon^+\, B \mid B) \in P}\ \varepsilon^+\text{-INT}$$

Finally, to sample $\sigma \sim \Delta_q(\underset{\sim}{\sigma})$, we first interleave $\underset{\sim}{\sigma}$ as $\underset{\sim}{\sigma}_\varepsilon$ (see Lemma **??**), then enumerate hole templates $\mathrm{H}(\underset{\sim}{\sigma}_\varepsilon, i) = \sigma_{1\ldots i-1}\_\_\sigma_{i+1\ldots n}$ for each $i \in \cdot \in \left\{ {n \atop d} \right\}$ and $d \in 1 \ldots q$, then solve for $\tilde{\sigma} \in \mathrm{H}(\underset{\sim}{\sigma}_\varepsilon, i)$ satisfying $[S \in \Lambda^*_{\tilde{\sigma},\mathcal{G}}] \Leftrightarrow [\tilde{\sigma} \in \mathcal{L}(\mathcal{G})]$. If $\boldsymbol{\sigma} \coloneqq \mathrm{H}(\underset{\sim}{\sigma}_\varepsilon, i)$ is nonempty, then each edit from each patch in each $\tilde{\sigma} \in \boldsymbol{\sigma}$ will match one of the following patterns, covering all three Levenshtein edits:

$$\text{Deletion} = \left\{ \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots \quad \gamma_{1,2} = \varepsilon \right.$$

$$\text{Substitution} = \begin{cases} \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \gamma_1 \neq \varepsilon \wedge \gamma_2 = \varepsilon \\ \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \gamma_1 = \varepsilon \wedge \gamma_2 \neq \varepsilon \\ \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \{\gamma_1, \gamma_2\} \cap \{\varepsilon, \sigma_i\} = \varnothing \end{cases}$$

$$\text{Insertion} = \begin{cases} \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \gamma_1 = \sigma_i \wedge \gamma_2 \notin \{\varepsilon, \sigma_i\} \\ \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \gamma_1 \notin \{\varepsilon, \sigma_i\} \wedge \gamma_2 = \sigma_i \\ \ldots \sigma_{i-1}\, \gamma_1\, \gamma_2\, \sigma_{i+1} \ldots & \gamma_{1,2} = \sigma_i \end{cases}$$

Although complete with respect to $\Delta_q(\underset{\sim}{\sigma})$, this approach can produce patches containing more Levenshtein edits than are strictly necessary to repair $\underset{\sim}{\sigma}$. To ensure patches are both minimal and syntactically valid, we first introduce a simple technique to minimize the repairs in §1.7. By itself, uniformly sampling minimal repairs $\tilde{\sigma} \sim \Delta_q(\underset{\sim}{\sigma}) \cap \mathcal{L}(\mathcal{G})$ is sufficient but can be quite time-consuming, as we empirically show in §**??**. To further reduce sample complexity and enable real-time repairs, we will then introduce a more efficient density estimator based on adaptive resampling (§1.8).

1.7 Patch minimization

Suppose we have a string, `a ( b`, and discover the patch, $\tilde{\sigma}$ = `( a + b )`. Although $\tilde{\sigma}$ is syntactically admissible, it is not minimal. To minimize a patch, we consider the set of all of its constituent subpatches, namely, `( a + b`, `( a ( b )`, `a + b )`, `( a ( b`, `a + b`, and `a ( b )`, then retain only the smallest syntactically valid instance(s) by Levenshtein distance. This forms a so-called *patch powerset*, which can be lazily enumerated from the top-down using the Maculay representation, after which we take all valid elements from the lowest level containing at least one admissible element, i.e., `a + b` and `a ( b )`. When patches are very large, minimization can be used in tandem with the delta debugging technique [6] to first simplify contiguous edits, then apply the patch powerset construction. Minimization is often useful for estimating the language edit distance: given a single valid repair of arbitrary size, minimization lets us quickly approximate an upper-bound on $\Delta(\sigma, \ell)$.

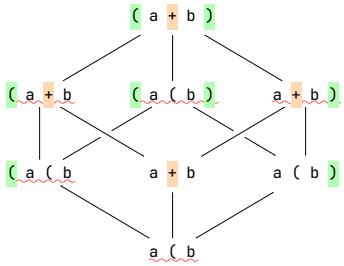

Fig. 1: Patch powerset of $\tilde{\sigma}$ = `( a + b )`.

1.8 Probabilistic reachability

Since there are $\Sigma_{d=1}^{q} \binom{n}{d}$ total hole templates, each with $|\Sigma_\varepsilon|^d$ individual edits to check, if n and q are large, this space can be slow to exhaustively search and a uniform prior may be highly sample-inefficient. Furthermore, naïvely sampling $\sigma \sim \Delta_q(\sigma)$ is likely to produce a large number of unnatural edits and converge poorly on $\Delta_q(\sigma) \cap \mathcal{L}(\mathcal{G})$. To rapidly rank and render relevant repair recommendations, we prioritize candidate edits according to the following six-step procedure:

1. Draw samples $\hat{\sigma} \sim \Delta_q(\sigma)$ without replacement using §1.6 with leapfrog parallelization.
2. Score by perplexity $PP(\hat{\sigma})$ using a pretrained variable-order Markov chain (VOMC) [4].
3. Resample using a concurrent variant of the A-Res [2] online weighted reservoir sampler.
4. Filter Levenshtein edits by admissibility with respect to the grammar, i.e., $[\hat{\sigma} \in \mathcal{L}(\mathcal{G})]$.
5. Minimize and store admissible repairs to a replay buffer, $\mathcal{Q} \leftarrow \tilde{\sigma}$, ranked by perplexity.

6. Repeat steps (1)-(5), alternately sampling from the LFSR/VOMC-reweighted online resevoir sampler with probability ϵ or stochastically resampled $\mathcal{Q}$ with probability $(1-\epsilon)$, where ϵ decreases from 1 to 0 according to a stepwise schedule relative to the time remaining.

Initially, the replay buffer $\mathcal{Q}$ is empty and repairs are sampled uniformly without replacement from the Levenshtein ball, $\Delta_q(\underset{\sim}{\sigma})$. As time progresses, $\mathcal{Q}$ is gradually populated with admissible repairs and resampled with increasing probability, allowing the algorithm to initially explore, then exploit the most promising candidates. This is summarized in Algorithm 1 which is run in parallel across all available CPU cores.

Algorithm 1 Probabilistic reachability

Require: $\mathcal{G}$ grammar, $\underset{\sim}{\sigma}$ broken string, p process ID, c total CPU cores, t_{total} timeout.
$\mathcal{Q} \leftarrow \varnothing, \mathcal{R} \leftarrow \varnothing, \epsilon \leftarrow 1, i \leftarrow 0, Y \sim \mathbb{Z}_2^m, t_0 \leftarrow t_{\text{now}}$ ▷ Initialize replay buffer $\mathcal{Q}$ and reservoir $\mathcal{R}$.
repeat
  if $\mathcal{Q} = \varnothing$ or **Rand**$(0, 1) < \epsilon$ **then**
    $\tilde{\sigma} \leftarrow \varphi^{-1}\left(\langle \kappa, \rho \rangle^{-1}(U^{ci+p}Y), \underset{\sim}{\sigma}\right), i \leftarrow i+1$ ▷ Sample WoR using the leapfrog method.
  else
    $\tilde{\sigma} \sim \mathcal{Q} +$ **Noise**$(\mathcal{Q})$ ▷ Sample replay buffer with additive noise.
  end if
  $\mathcal{R} \leftarrow \mathcal{R} \cup \{\tilde{\sigma}\}$ ▷ Insert repair candidate $\tilde{\sigma}$ into reservoir $\mathcal{R}$.
  if $\mathcal{R}$ is full **then**
    $\tilde{\sigma} \leftarrow \text{argmin}_{\tilde{\sigma} \in \mathcal{R}}\, PP(\tilde{\sigma})$ ▷ Select lowest perplexity repair candidate.
    if $\tilde{\sigma} \in \mathcal{L}(\mathcal{G})$ **then**
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\tilde{\sigma}\}$ ▷ Insert successful repair into replay buffer.
    end if
    $\mathcal{R} \leftarrow \mathcal{R} \setminus \{\tilde{\sigma}\}$ ▷ Remove checked sample from the reservoir.
  end if
  $\epsilon \leftarrow$ **Schedule**$((t_{\text{now}} - t_0)/t_{\text{total}})$ ▷ Update exploration/exploitation rate.
until t_{total} elapses.
return $\tilde{\sigma} \in \mathcal{Q}$ ranked by $PP(\tilde{\sigma})$.

We would prefer hole templates likely to yield repairs that are (1) admissible (i.e., grammatically correct) and (2) plausible (i.e., likely to have been written by a human author). To do so, we draw holes and rank admissible repairs using a probabilistic distance metric over $\Delta_q(\underset{\sim}{\sigma})$. For example, suppose we are given an invalid string, $\underset{\sim}{\sigma}_\varepsilon : \Sigma^{90}$ and $\mathcal{Q} \subseteq [0, |\sigma_\varepsilon|) \times \Sigma_\varepsilon^q$, a distribution over previously successful edits, which we can use to localize admissible repairs. By marginalizing onto $\underset{\sim}{\sigma}_\varepsilon$, the distribution $\mathcal{Q}(\underset{\sim}{\sigma}_\varepsilon)$ could take the form depicted in Fig. 2.

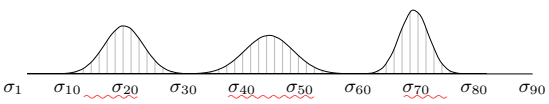

Fig. 2: The distribution $\mathcal{Q}$, projected onto the invalid string, suggests edit locations likely to yield admissible repairs, from which we draw subsets of size d.

More specifically in our setting, we want to sample from a discrete product space that factorizes into (1) the specific edit locations (e.g., informed by caret position, historical edit locations, or a static analyzer), (2) probable completions (e.g., from a Markov chain or neural language model) and (3) an accompanying *cost model*, $C : (\Sigma^* \times \Sigma^*) \to \mathbb{R}$, which may be any number of suitable distance metrics, such as language edit distance, finger travel distance on a physical keyboard, weighted Levenshtein distance, or stochastic contextual edit distance [1] in the case of probabilistic edits. Our goal then, is to discover repairs which minimize $C(\underset{\sim}{\sigma}, \tilde{\sigma})$, subject to the given grammar and latency constraints.

1.9 Ranking

Since the number of solutions can be very large, we can use a language model to rank the results maximizing likelihood, or minimizing perplexity, subject to the constraints. This ranking can be used to guide the propagation, sample the choice function, sample hole locations or as a post-processing step after a fixed timeout has expired.

References

1. Cotterell, R., Peng, N., Eisner, J.: Stochastic contextual edit distance and probabilistic FSTs. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. vol. 2 (Short Papers), pp. 625–630. Association for Computational Linguistics, Baltimore, Maryland (June 2014)
2. Efraimidis, P.S.: Weighted random sampling over data streams. Algorithms, Probability, Networks, and Games: Scientific Papers and Essays Dedicated to Paul G. Spirakis on the Occasion of His 60th Birthday pp. 183–195 (2015)
3. Knuth, D.E.: Generating All Combinations and Partitions, pp. 5–6. Addison-Wesley (2005)
4. Schulz, M.H., Weese, D., Rausch, T., Döring, A., Reinert, K., Vingron, M.: Fast and adaptive variable order markov chain construction. In: Algorithms in Bioinformatics: 8th International Workshop, WABI 2008, Karlsruhe, Germany, September 15-19, 2008. Proceedings 8. pp. 306–317. Springer (2008)
5. Szudzik, M.: An elegant pairing function. In: Special NKS 2006 Wolfram Science Conference. pp. 1–12 (2006)
6. Zeller, A.: Isolating cause-effect chains from computer programs. ACM SIGSOFT Software Engineering Notes **27**(6), 1–10 (2002)