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 error correction.

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 $\sqsubseteq : \Sigma^n \times \underline{\Sigma}^n$ 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 start with some background, 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 $\hat{\sigma}_r := \{w \mid (w \to \sigma_r) \in P\}$, then construct a matrix with nonterminals on the superdiagonal representing each token, $M_0[r+1=c](G', \sigma) := \hat{\sigma}_r$ and solve for the fixpoint $M_{i+1} = M_i + M_i^2$,

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

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, $[S \in V_{0,3}] \Leftrightarrow [V_{0,3,3} = \blacksquare] \Leftrightarrow [A(\sigma) \neq \varnothing]$. 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 $V_{0,2}, V_{1,3}$ 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 solve for the corner entry $V_{0,3}$ by taking the bitwise dot product between the first row and last column, yielding:

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

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 $\{1\} \times \hat{\sigma}_r^{-1}(O) \times \hat{\sigma}_r^{-1}(N)$ 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.

The question naturally arises, where should one put the holes? One solution is to interleave ε between every token as $\sigma_\varepsilon := (\varepsilon^c \sigma_i)_{i=1}^n \varepsilon^c$, augment the grammar to admit ε^+, then sample holes without replacement from all possible locations. Below we illustrate 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. `d = sum([foo(+i) for i in vals])`
8. `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 (7), then reduced

to its minimal form in step (8), and sampling is repeated until all possibilities are exhausted or a predetermined timeout expires.

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 3), then enumerate hole templates $\mathrm{H}(\underset{\sim}{\sigma}_\varepsilon, i) = \sigma_{1\ldots i-1}\ \_\ \_\ \sigma_{i+1\ldots n}$ for each $i \in \cdot \in \binom{n}{d}$ 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:

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

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

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

1.4 Bar-Hillel Construction

Manually generating the edits is more controllable way to synthesize edits, but can be unnecessarily expensive if the goal is to synthesize all edits within a fixed edit distance. The second approach is more efficient, but requires generating a large grammar. We now describe the Bar-Hillel construction, which allows us to generate a grammar from a finite automaton, and then use the grammar to generate the edits without enumerating holes.

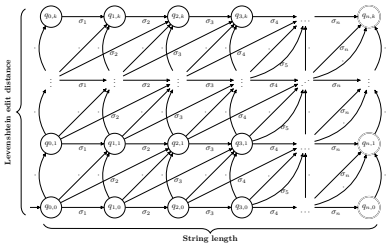
Definition 4. *A finite state automata is a tuple $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$, where Q is a finite set of states, Σ is a finite alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition function, and $I, F \subseteq Q$ are the set of initial and final states, respectively.*

Lemma 1. *For any context-free language ℓ and finite state automata α, there exists a context-free grammar $G^\cap$ such that $\mathcal{L}(G^\cap) = \ell \cap \mathcal{L}(\alpha)$. See Bar-Hillel [1].*

Beigel and Gasarch [2] provide one explicit way to construct $G^{\cap}$:

$$\frac{q \in I \quad r \in F}{\big(S \to qSr\big) \in P^{\cap}} \qquad \frac{(q \xrightarrow{a} r) \in \delta}{\big(qar \to a\big) \in P^{\cap}} \qquad \frac{(w \to xz) \in P \qquad p, q, r \in Q}{\big(pwr \to (pxq)(qzr)\big) \in P^{\cap}}$$

Conveniently, the Levenshtein ball is recognized by a nondeterministic finite automata (NFA). From Lemma 1, we know the intersection of any context-free language and NFA is context-free, and therefor we can construct a single context-free grammar $G^{\cap}$ which recognizes and generates the admissible set.



NIA

CFG

$$S \Rightarrow \{\cdot \in Q \mid \delta(\cdot, q_{n,0}) \leq k\}$$
$$+ \Rightarrow \{\cdot \in \Sigma\}$$
$$\big\{q_{i,j} \Rightarrow \{q_{i,j-1}+\} \mid i,j \in [0,n] \times [1,k]\big\}$$
$$\big\{q_{i,j} \Rightarrow \{q_{i-1,j-1}+\} \mid i,j \in [1,n] \times [1,k]\big\}$$
$$\big\{q_{i,j} \Rightarrow \{q_{i-1,j}\sigma_i\} \mid i,j \in [1,n] \times [0,k]\big\}$$
$$\big\{q_{i,j} \Rightarrow \{q_{i-2,j-1}\sigma_i\} \mid i,j \in [2,n] \times [1,k]\big\}$$

Fig. 1: Levenshtein reachability from Σ^n can be described as an NFA, or a CFG.

Alternatively, this transition system can be viewed as a kind of proof system.

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

$$\frac{s = \sigma_i \quad i \in [1,n] \quad j \in [0,k]}{(q_{i-1,j} \xrightarrow{s} q_{i,j}) \in \delta} \rightarrow \qquad \frac{s = \sigma_i \quad i \in [2,n] \quad j \in [1,k]}{(q_{i-2,j-1} \xrightarrow{s} q_{i,j}) \in \delta} \nearrow\!\!\rightarrow$$

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

These rewrite rules can generate a large grammar whose cardinality is approximated by $|P^{\cap}| = |I||F| + |\delta| + |P||Q|^3$. Since it is so large, materializing the grammar directly can be expensive, however instead of materializing it directly, we can lazily enumerate its inhabitants by performing a kind of proof search.

1.5 Semiring Algebras

There are a number of alternate semirings which can be used to solve for $A(\sigma)$. A first approach propagates the values from the bottom-up, while mapping nonterminals to lists of strings. Letting $D = V \to \mathcal{P}(\Sigma^*)$, we define $\oplus, \otimes : D \times D \to D$. Initially, we construct $M_0[r+1=c] = \hat{\sigma}_r = p(\sigma_r)$ as follows:

$$p(s:\Sigma) \mapsto \{v \mid (v \to s) \in P\} \text{ and } p(\_) := \bigcup_{s \in \Sigma} p(s) \tag{17}$$

Like the recognizer defined in § 1.2, $p(\cdot)$ constructs elements of the superdiagonal, then we compute the fixpoint using the algebra:

$$X \oplus Z \mapsto \big\{v \to \big(X(v) \cup Z(v)\big) \mid v \in V\big\} \tag{18}$$

$$X \otimes Z \mapsto \bigoplus_{w,x,z} \big\{w \to (l+r) \mid (w \to xz) \in P, l \in X(x), r \in Z(x)\big\} \tag{19}$$

After the fixpoint M_∞ is attained, the solutions can be read off via $\Lambda^*_\sigma[S]$. The issue here is an exponential growth in cardinality when eagerly computing the Cartesian product, which becomes impractical for even small strings.

This encoding can be made more compact by propagating an algebraic data type $\mathbb{T}_3 = (V \cup \Sigma) \rightharpoonup \mathbb{T}_2$ where $\mathbb{T}_2 = (V \cup \Sigma) \times (\mathbb{N} \rightharpoonup \mathbb{T}_2 \times \mathbb{T}_2)$[4]. Morally, we can think of $\mathbb{T}_2$ as an implicit set of possible trees sharing the same root, and $\mathbb{T}_3$ as a dictionary of possible $\mathbb{T}_2$ values indexed by possible roots, given by a specific CFG under a finite-length porous string. We construct $\hat{\sigma}_r = \dot{p}(\sigma_r)$ as follows:

$$\dot{p}(s:\Sigma) \mapsto \Big\{\mathbb{T}_2\big(v, \big[\langle \mathbb{T}_2(s), \mathbb{T}_2(\varepsilon)\rangle\big]\big) \mid (v \to s) \in P\Big\} \text{ and } \dot{p}(\_) := \bigoplus_{s \in \Sigma} p(s) \tag{20}$$

We compute the fixpoint M_∞ by defining $\oplus, \otimes : \mathbb{T}_3 \times \mathbb{T}_3 \to \mathbb{T}_3$ as follows:

$$X \oplus Z \mapsto \big\{\mathbb{T}_2(k, Q_x \cup Q_z) \mid (k,Q)_X \bowtie_k (k,Q)_Z\big\} \tag{21}$$

$$X \otimes Z \mapsto \bigoplus_{w,x,z} \Big\{\mathbb{T}_2\big(w, \{\langle X \circ x, Z \circ z\rangle\}\big) \mid (w \to xz) \in P, x \in \pi_1(X), z \in \pi_1(Z)\Big\} \tag{22}$$

Decoding trees from $\Lambda^*_\sigma[S] : \mathbb{T}_2$ becomes a straightforward matter of enumeration using a recursive choice function that emits a sequence of binary trees generated by the CFG. We define this construction more precisely in § 1.6.

In our experiments, we provide a comparison of the performance of the SAT algebra and these two semirings, evaluated on a dataset of Python statements.

[4] Hereinafter, given a concrete $T : \mathbb{T}_2$, we shall refer to $\pi_1(T), \pi_2(T)$ as $\texttt{root}(T)$ and $\texttt{children}(T)$ respectively.

1.6 A Pairing Function for Breadth-Bounded Binary Trees

The type $\mathbb{T}_2$ of all possible trees that can be generated by a CFG in Chomksy Normal Form is identified by a recurrence relation:

$$L(p) = 1 + pL \qquad\qquad P(a) = 1 + aL(P(a)^2) \tag{23}$$

The number of binary trees inhabiting a single instance of $\mathbb{T}_2$ is sensititive to the number of nonterminals and rule expansions in the grammar. To obtain the total number of trees with breadth n, we can take the intersection between a CFG and the regular language, $\mathcal{L}(G^{\cap}) := \mathcal{L}(\mathcal{G}) \cap \Sigma^n$, abstractly parse the string containing all holes, let $T = \Lambda^*_{\underline{\sigma}}[S]$, and compute the total number of trees using the following recurrence:

$$|T : \mathbb{T}_2| \mapsto \begin{cases} 1 & \text{if } T \text{ is a leaf,} \\ \sum_{\langle T_1, T_2 \rangle \in \texttt{children}(T)} |T_1| \cdot |T_2| & \text{otherwise.} \end{cases} \tag{24}$$

To sample all trees in a given $T : \mathbb{T}_2$ uniformly without replacement, we first define a pairing function $\varphi^{-1} : \mathbb{T}_2 \to \mathbb{Z}_{|T|} \to \texttt{BTree}$ as follows:

$$\varphi^{-1}(T : \mathbb{T}_2, i : \mathbb{Z}_{|T|}) \mapsto \begin{cases} \Big\langle \texttt{BTree}\big(\texttt{root}(T)\big), i \Big\rangle & \text{if } T \text{ is a leaf,} \\ \text{Let } b = |\texttt{children}(T)|, & \\ \quad q_1, r = \big\langle \lfloor \frac{i}{b} \rfloor, i \pmod b \big\rangle, & \\ \quad lb, rb = \texttt{children}[r], & \\ \quad T_1, q_2 = \varphi^{-1}(lb, q_1), & \\ \quad T_2, q_3 = \varphi^{-1}(rb, q_2) \text{ in} & \\ \Big\langle \texttt{BTree}\big(\texttt{root}(T), T_1, T_2\big), q_3 \Big\rangle & \text{otherwise.} \end{cases} \tag{25}$$

Then, instead of sampling trees, we can simply sample integers uniformly without replacement from $\mathbb{Z}_{|T|}$ using the construction defined in 1.8, and lazily decode them into trees.

1.7 Complexity

Let us consider some loose bounds on the complexity of BCFLR. To do, we first consider the complexity of computing language-edit distance, which is a lower-bound on BCFLR complexity.

Definition 5. *Language edit distance (LED) is the problem of computing the minimum number of edits required to transform an invalid string into a valid one, where validity is defined as containment in a context-free language, $\ell : \mathcal{L}$, i.e., $\Delta^*(\underset{\sim}{\sigma}, \ell) := \min_{\sigma \in \ell} \Delta(\underset{\sim}{\sigma}, \sigma)$, and Δ is the Levenshtein distance. LED is known to have subcubic complexity [3].*

We seek to find the set of strings S such that $\forall \tilde{\sigma} \in S, \Delta(\underset{\sim}{\sigma}, \tilde{\sigma}) \leq q$, where q is the maximum number of edits greater than or equal to the language edit distance. We call this set the *Levenshtein ball* of $\underset{\sim}{\sigma}$ and denote it $\Delta_q(\underset{\sim}{\sigma})$. Since $1 \leq \Delta^*(\underset{\sim}{\sigma}, \ell) \leq q$, we have $1 \leq q$. We now consider an upper bound on $\Delta^*(\underset{\sim}{\sigma}, \ell)$, i.e., the greatest lower bound on q such that $\Delta_q(\underset{\sim}{\sigma}) \cap \ell \neq \varnothing$.

Lemma 2. *For any nonempty language $\ell : \mathcal{L}$ and invalid string $\underset{\sim}{\sigma} : \Sigma^n \cap \bar{\ell}$, there exists an $(\tilde{\sigma}, m)$ such that $\tilde{\sigma} \in \ell \cap \Sigma^m$ and $0 < \Delta(\underset{\sim}{\sigma}, \ell) \leq \max(m, n) < \infty$.*

Proof. Since ℓ is nonempty, it must have at least one inhabitant $\sigma \in \ell$. Let $\tilde{\sigma}$ be the smallest such member. Since $\tilde{\sigma}$ is a valid sentence in ℓ, by definition it must be that $|\tilde{\sigma}| < \infty$. Let $m \coloneqq |\tilde{\sigma}|$. Since we know $\underset{\sim}{\sigma} \notin \ell$, it follows that $0 < \Delta(\underset{\sim}{\sigma}, \ell)$. Let us consider two cases, either $\tilde{\sigma} = \varepsilon$, or $0 < |\tilde{\sigma}|$:

- If $\tilde{\sigma} = \varepsilon$, then $\Delta(\underset{\sim}{\sigma}, \tilde{\sigma}) = n$ by full erasure of $\underset{\sim}{\sigma}$, or
- If $0 < m$, then $\Delta(\underset{\sim}{\sigma}, \tilde{\sigma}) \leq \max(m, n)$ by overwriting.

In either case, it follows $\Delta(\underset{\sim}{\sigma}, \ell) \leq \max(m, n)$ and ℓ is always reachable via a finite nonempty set of Levenshtein edits, i.e., $0 < \Delta(\underset{\sim}{\sigma}, \ell) < \infty$.

Let us now consider the maximum growth rate of the *admissible set*, $A \coloneqq \Delta_q(\underset{\sim}{\sigma}) \cap \ell$, as a function of q and n. Let $\bar{\ell} \coloneqq \{\underset{\sim}{\sigma}\}$. Since $\bar{\ell}$ is finite and thus regular, $\ell = \Sigma^* \setminus \{\underset{\sim}{\sigma}\}$ is regular by the closure of regular languages under complementation, and thus context-free a fortiori. Since ℓ accepts every string except $\underset{\sim}{\sigma}$, it represents the worst CFL in terms of asymptotic growth of A.

Lemma 3. *The complexity A is upper bounded by $\mathcal{O}\left(\sum_{c=1}^{q} \binom{cn+n+c}{c} (|\Sigma| + 1)^c\right)$.*

Proof. We can overestimate the size of A by considering the number of unique ways to insert, delete, or substitute c terminals into a string $\underset{\sim}{\sigma}$ of length n. This can be overaproximated by interleaving ε^c around every token, i.e., $\underset{\sim}{\sigma}_\varepsilon \coloneqq (\varepsilon^c \underset{\sim}{\sigma}_i)_{i=1}^n \varepsilon^c$, where $|\underset{\sim}{\sigma}_\varepsilon| = cn + n + c$, and only considering substitution. We augment $\Sigma_\varepsilon \coloneqq \Sigma \cup \{\varepsilon\}$ so that deletions and insertions may be treated as special cases of substitution. Thus, we have $cn + n + c$ positions to substitute ($|\Sigma_\varepsilon|$) tokens, i.e., $\binom{cn+n+c}{c}|\Sigma_\varepsilon|^c$ ways to edit $\underset{\sim}{\sigma}_\varepsilon$ for each $c \in [1, q]$. This upper bound is not tight, as overcounts many identical edits w.r.t. $\underset{\sim}{\sigma}$. Nonetheless, it is sufficient to show $|A| < \sum_{c=1}^{q} \binom{cn+n+c}{c}|\Sigma_\varepsilon|^c$.

We note that the above bound applies to all strings and languages, and relates to the Hamming bound on $H_q(\underset{\sim}{\sigma}_\varepsilon)$, which only considers substitutions. [5] In practice, much tighter bounds may be obtained by considering the structure of ℓ and $\underset{\sim}{\sigma}$. For example, based on an empirical evaluation from a dataset of human errors and repairs in Python code snippets ($|\Sigma| = 50, |\underset{\sim}{\sigma}| < 40, \Delta(\underset{\sim}{\sigma}, \ell) \in [1, 3]$), we estimate the *filtration rate*, i.e., the density of the admissible set relative to the Levenshtein ball, $D = |A|/|\Delta_q(\underset{\sim}{\sigma})|$ to have empirical mean $E_\sigma[D] \approx 2.6 \times 10^{-4}$, and variance $\mathrm{Var}_\sigma[D] \approx 3.8 \times 10^{-7}$.

[5] This reflects our general approach, which builds a surjection from the interleaved Hamming ball onto the Levenshtein ball.

1.8 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, $F : (\mathcal{G} \times \Sigma^n) \to \{\Sigma^n\} \subseteq \mathcal{L}(\mathcal{G})$, 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, $\boldsymbol{\sigma} \sim \Sigma^{n \pm q} \cap \Delta_q(\sigma)$ uniformly across the Levenshtein q-ball centered on σ, i.e., the space of all admissible edits with Levenshtein distance $\leq q$.

To implement this strategy, we first construct a surjection $\varphi^{-1} : \mathbb{Z}_2^m \twoheadrightarrow \Delta_q(\sigma)$ from bitvectors to Levenshtein edits over σ, Σ, 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 and ρ will index Σ^k tuples, but is slightly more tricky to define. To maximize compactness, there is an elegant pairing function by Szudzik [7], which enumerates concentric square shells over $\mathbb{N}^2$ 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 $m := \lceil \log_b \binom{n}{k} |\Sigma_\varepsilon|^k \rceil$. 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 coefficients $C_{1 \ldots m}$ and semiring operators

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

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

$\oplus := + \pmod 2, \otimes := \wedge, \top := 1, \circ := 0$. 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 surjection from $\mathbb{Z}_2^m$ onto the Levenshtein 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 $b^{-m}\binom{n}{d}|\Sigma_\varepsilon|^d$ can be slightly improved with a more suitable base b, 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 leapfrogging, i.e., given p independent processors, each one p_j can independently check $[\varphi^{-1}(\langle\kappa, \rho\rangle^{-1}(\mathbf{R}_i), \varrho) \in \mathcal{L}(\mathcal{G})]$ where $p_j \equiv i \pmod{|p|}$. This procedure linearly scales with the total processors, exhaustively searching $\Delta_q(\varrho)$ in $|p|^{-1}$ of the time required by a single processor, or alternately drawing $|p|$ times as many samples in the same time.

Although complete with respect to $\Delta_q(\varrho)$, this approach can produce patches containing more Levenshtein edits than are strictly necessary to repair ϱ. To ensure patches are both minimal and syntactically valid, we first introduce a simple technique to minimize the repairs in §1.9. By itself, uniformly sampling minimal repairs $\tilde{\sigma} \sim \Delta_q(\varrho) \cap \mathcal{L}(\mathcal{G})$ is sufficient but can be quite time-consuming. To further reduce sample complexity and enable real-time repairs, we will then introduce a more efficient density estimator based on adaptive resampling (§1.10).

1.9 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, after which we take all valid strings from the lowest level containing at least one valid string, i.e., `a + b` and `a ( b )`. When patches are very large, minimization can be used in tandem with the delta debugging technique [8] 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 $\Delta(\varrho, \ell)$.

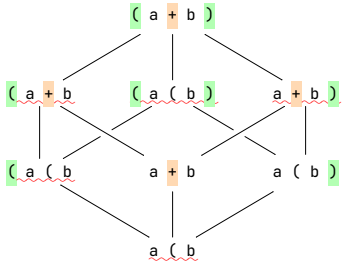


Fig. 2: $\tilde{\sigma}$ = `( a + b )`.

1.10 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 $\sigma \sim \Delta_q(\underset{\sim}{\sigma})$ is likely to produce a large number of unnatural edits and converge poorly on $\Delta_q(\underset{\sim}{\sigma}) \cap \mathcal{L}(\mathcal{G})$. To rapidly rank and render relevant repair recommendations, we prioritize candidate edits according to the following procedure.

(1) Draw samples $\hat{\sigma} \sim \Delta_q(\underset{\sim}{\sigma})$ without replacement using §1.8 with leapfrog parallelization. (2) Score by perplexity $PP(\hat{\sigma})$ using a pretrained variable-order Markov chain (VOMC) [6]. (3) Resample using a concurrent variant of the A-Res [5] 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.

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

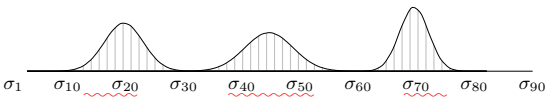


Fig. 3: The distribution $\mathcal{Q}$, projected onto σ, suggests edit locations likely to yield admissible repairs, from which we draw subsets of size d.

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. Marginalizing onto $\underset{\sim}{\sigma_\varepsilon}$, the distribution $\mathcal{Q}(\underset{\sim}{\sigma_\varepsilon})$ may take the form shown in Fig. 3.

More specifically, we want to sample from a discrete product space that factorizes into (1) the edit locations (e.g., informed by caret position, historical edit locations, etc.), (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, weighted Levenshtein distance, or stochastic contextual edit distance [4] in the case of probabilistic edits. Our goal then, is to discover repairs minimizing $C(\underset{\sim}{\sigma}, \tilde{\sigma})$, subject to the given grammar and latency constraints.

References

1. Bar-Hillel, Y., Perles, M., Shamir, E.: On formal properties of simple phrase structure grammars. Sprachtypologie und Universalienforschung **14**, 143–172 (1961)
2. Beigel, R., Gasarch, W.: 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`
3. Bringmann, K., Grandoni, F., Saha, B., Williams, V.V.: Truly subcubic algorithms for language edit distance and rna folding via fast bounded-difference min-plus product. SIAM Journal on Computing **48**(2), 481–512 (2019)
4. 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)
5. 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)
6. 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)
7. Szudzik, M.: An elegant pairing function. In: Special NKS 2006 Wolfram Science Conference. pp. 1–12 (2006)
8. Zeller, A.: Isolating cause-effect chains from computer programs. ACM SIGSOFT Software Engineering Notes **27**(6), 1–10 (2002)