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 the language is sound and complete with respect to the programming langauge 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

We will first give some background on Brozozowski differentiation. Let us 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:

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

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:

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

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

[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.

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

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

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 $\sigma \in \mathcal{L}(e)$:*

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

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

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 constructing a parse chart that represents the intersection.

Theorem 2.7. *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 sketch. 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 $\texttt{choose}(S_\cap)$. □

2.2 Combinatorics

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

Definition 2.8 (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 2.9 (Enumeration). *To enumerate, invoke* $\bigcup_{i=0}^{|R|}\{\mathit{enum}(R,i)\}$*:*

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

[2] Traditionally, there is a $\frac{1}{k!}$ factor to suppress exploding entries.

3 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.

3.1 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.

3.2 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.

3.3 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 3.1. *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.