1 Method

The syntax of most programming languages is context-free. Our proposed method is simple. We construct a context-free grammar representing the intersection between the langauge syntax and an automaton recognizing the Levenshtein ball of a given radius. Since CFLs are closed under intersection with regular languages, this is admissible. Three outcomes are possible:

1. $\mathcal{G}_\cap$ is empty, in which case there is no repair within the given radius. In this case, we simply increase the radius and try again.
2. $\mathcal{L}(\mathcal{G}_\cap)$ is small, in which case we simply enumerate all possible repairs. Enumeration is tractable for $\sim 80\%$ of the Python dataset in ≤ 90s.
3. $\mathcal{L}(\mathcal{G}_\cap)$ is too large to enumerate, so we sample from the intersection grammar $\mathcal{G}_\cap$. Sampling is necessary for $\sim 20\%$ of the Python dataset.

When ambiguous, we use an n-gram model to rank and return the top-k results by likelihood. This procedure is depicted in the flowchart below:

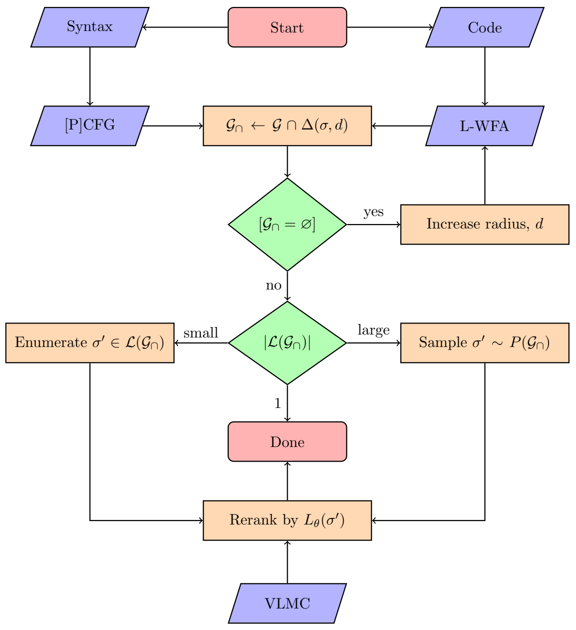
Syntax
Start
Code
[P]CFG
$\mathcal{G}_\cap \leftarrow \mathcal{G} \cap \Delta(\sigma, d)$
L-WFA
$[\mathcal{G}_\cap = \varnothing]$
yes
Increase radius, d
no
Enumerate $\sigma' \in \mathcal{L}(\mathcal{G}_\cap)$
small
$|\mathcal{L}(\mathcal{G}_\cap)|$
large
Sample $\sigma' \sim P(\mathcal{G}_\cap)$
1
Done
Rerank by $L_\theta(\sigma')$
VLMC

Figure 1: Flowchart of our proposed method.

1.1 The Nominal Levenshtein Automaton

Levenshtein edits are recognized by a certain kind of automaton, known as the Levenshtein automaton. Since the original approach used by Schultz and Mihov contains cycles and epsilon transitions, we propose a modified variant which is epsilon-free, acyclic and monotone. Furthermore, we use a nominal automaton, allowing for infinite alphabets. This considerably simplifies the langauge intersection.

1.2 The Bar-Hillel Construction

The Bar-Hillel construction is a general method for obtaining the context-free grammar representing the intersection of a context-free language and a regular language. We will now present the epsilon-free version of the Bar-Hillel construction used in our work.[1]

1.3 The Levenshtein-Bar-Hillel-Parikh Reduction

The standard BH construction applies to any CFL and REG language. While straightforward, the general method can generate hundreds of trillions of productions for moderately sized grammars and Levenshtein automata. Our method considerably simplifies this process by eliminating the need to materialize most of those productions, and is the key to making our approach tractable.

To achieve this, we precompute upper and lower Parikh bounds for every terminal and integer range of the string, which we call the Parikh map. This construction soundly overapproximates the minimum and maximum number of terminals that can be derived from a given nonterminal in a bounded-length string, and is used to prune the search space. We will now describe this reduction in detail.

[1]Clemente Pasti has a version of the BH construction that supports epsilon transitions, but is slightly more complicated.