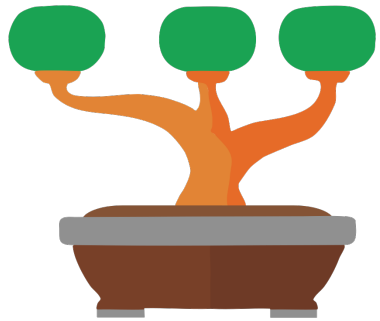
A Pragmatic Approach to Syntax Repair

Breandan Considine, Jin Guo, Xujie Si

McGill University, Mila IQIA

bre@ndan.co

August 3, 2024



Can you spot the error?

Original code	**Human repair**

`newlist = []`
`i = set([5, 3, 1)]`
`z = set([5, 0, 4)]`	

Can you spot the error?

Original code	**Human repair**

`newlist = []`
`i = set([5, 3, 1)]`
`z = set([5, 0, 4)]`
	`newlist = []`
	`i = set([5, 3, 1])`
	`z = set([5, 0, 4])`

Can you spot the error?

Original code	**Human repair**

`def average(values):`
`  if values == (1,2,3):`
`  return (1+2+3)/3`
`  else if values == (-3,2):`
`  return (-3+2+8-1)/4`	

Can you spot the error?

Original code	Human repair

`def average(values):`
`  if values == (1,2,3):`
`  return (1+2+3)/3`
`  else if values == (-3,2):`
`  return (-3+2+8-1)/4`
	`def average(values):`
	`  if values == (1,2,3):`
	`  return (1+2+3)/3`
	`  elif values == (-3,2):`
	`  return (-3+2+8-1)/4`

Can you spot the error?

Original code	**Human repair**

`import Global from Global`
`globalObj = Global()`
`print(str(globalObj.Test()))`	

Can you spot the error?

Original code	Human repair

`import Global from Global`
`globalObj = Global()`
`print(str(globalObj.Test()))`
	`from Global import Global`
	`globalObj = Global()`
	`print(str(globalObj.Test()))`

How many repairs could there possibly be?

Consider the following Python snippet, which contains a small syntax error:

```
def prepend(i, k, L=[])
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

How many repairs could there possibly be?

Consider the following Python snippet, which contains a small syntax error:

```
def prepend(i, k, L=[])
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

It can be fixed by appending a colon after the function signature, yielding:

```
def prepend(i, k, L=[]):
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

How many repairs could there possibly be?

Consider the following Python snippet, which contains a small syntax error:

```
def prepend(i, k, L=[])
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

It can be fixed by appending a colon after the function signature, yielding:

```
def prepend(i, k, L=[]):
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

Let us consider a slightly more ambiguous error: `v = df.iloc(5:, 2:)`. Assuming an alphabet of just a hundred lexical tokens, this statement has millions of two-token edits, yet only six are accepted by the Python parser:

How many repairs could there possibly be?

Consider the following Python snippet, which contains a small syntax error:

```
def prepend(i, k, L=[])
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

It can be fixed by appending a colon after the function signature, yielding:

```
def prepend(i, k, L=[]):
  n and [prepend(i - 1, k, [b] + L) for b in range(k)]
```

Let us consider a slightly more ambiguous error: `v = df.iloc(5:, 2:)`. Assuming an alphabet of just a hundred lexical tokens, this statement has millions of two-token edits, yet only six are accepted by the Python parser:

(1) `v = df.iloc(5:, 2,)`

(2) `v = df.iloc(5), 2()`

(3) `v = df.iloc(5[:, 2:])`

(4) `v = df.iloc(5:, 2:)`

(5) `v = df.iloc[5:, 2:]`

(6) `v = df.iloc(5[:, 2])`

On the virtues of pragmatism

Pragmatism: *a reasonable and logical way of solving problems that is based on dealing with specific situations instead of abstract theories.*

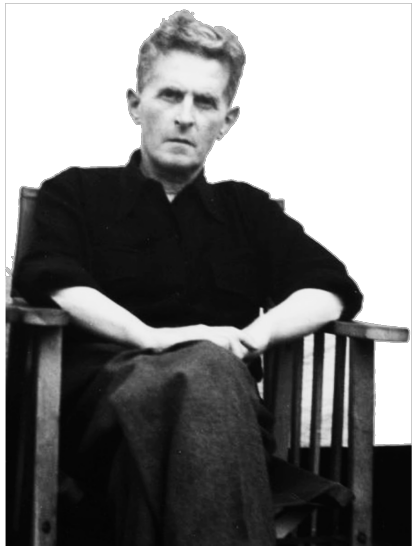
- Often framed as a compromise, *"Let's be pragmatic..."*
- Pragmatism is a principled approach to problem solving.
- Taken seriously, pragmatism is difficult because it requires modeling the needs of multiple stakeholders and balancing competing interests.
- Putting it into practice requires knowing your customer, understanding their workflow, considering the most appropriate solution out of a set of possible alternatives.

"*What is the use of studying philosophy if all that it does for you is to enable you to talk about some abstruse questions of logic and does not improve your thinking about the important questions of everyday life?*"

Ludwig Wittgenstein, 1889–1951

On the virtues of pragmatism

- Pioneered in the 19th century by Peirce, James, Dewey, et al.
- Wittgenstein was a pragmatist, early work on language games.
- Pragmatism is a philosophy of language that emphasizes the role of intent in human communication.
- Language is a tool for communication, not just a arbitrary set of rules.
- Must actively imagine the mindset of the speaker, not just the literal meaning of their words.
- Language is a bit like a game whose goal is to understand the speaker's intent.
- Assume a proficient speaker, who is trying to communicate something meaningful.



From Error-Correcting Codes to Correcting Coding Errors

- Error-correcting codes are a well-studied topic in information theory used to detect and correct errors in data transmission.
- Introduces parity bits to detect and correct transmission errors assuming a certain noise model (e.g., Hamming distance).
- Like ECCs, we also assume a certain noise model (Levenshtein distance) and error tolerance (n-lexical tokens).
- Instead of injecting parity bits, we use the grammar and mutual information between tokens to detect and correct errors.
- Unlike ECCs, we do not assume a unique solution, but a set of admissible solutions ranked by statistical likliehood.

"*Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?*""

Richard Hamming, 1915-1998

Syntax repair as a language game

- Imagine a game between two players, *Editor* and *Author*.
- They both see the same grammar, $\mathcal{G}$ and invalid string $\underset{\sim}{\sigma} \notin \mathcal{L}(\mathcal{G})$.
- Author moves by modifying $\underset{\sim}{\sigma}$ to produce a valid string, $\sigma \in \mathcal{L}(\mathcal{G})$.
- Editor moves continuously, sampling a set $\tilde{\boldsymbol{\sigma}} \subseteq \mathcal{L}(\mathcal{G})$.
- As soon as Author repairs $\underset{\sim}{\sigma}$, the turn immediately ends.
- Neither player sees the other's move(s) before making their own.
- If Editor anticipates Author's move, i.e., $\sigma \in \tilde{\boldsymbol{\sigma}}$, they both win.
- If Author surprises Editor with a valid move, i.e., $\sigma \notin \tilde{\boldsymbol{\sigma}}$, Author wins.
- We may consider a refinement where Editor wins in proportion to the time taken to anticipate Author's move.

Can you spot the error?

Original code	**Valid repairs**

Can you spot the error?

Original code	Valid repairs
( ))	

Can you spot the error?

Original code	**Valid repairs**
( ))	( )

Can you spot the error?

Original code	**Valid repairs**
( ))	( ) ( ) ( )

Can you spot the error?

Original code	**Valid repairs**
( ))	( ) ( ) ( ) (( ))

Problem Statement: Validity and naturalness

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

Definition (Reachable edits)

Given a CFL, ℓ, and an invalid string, $\underset{\sim}{\sigma} : \ell^{\complement}$, find every valid string reachable within d edits of $\underset{\sim}{\sigma}$, i.e., letting Δ be the Levenshtein metric and $L(\underset{\sim}{\sigma}, d) = \{\sigma' \mid \Delta(\underset{\sim}{\sigma}, \sigma') \leq d\}$ be the edit ball, we seek $A = L(\underset{\sim}{\sigma}, d) \cap \ell$.

Definition (Ranked repair)

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

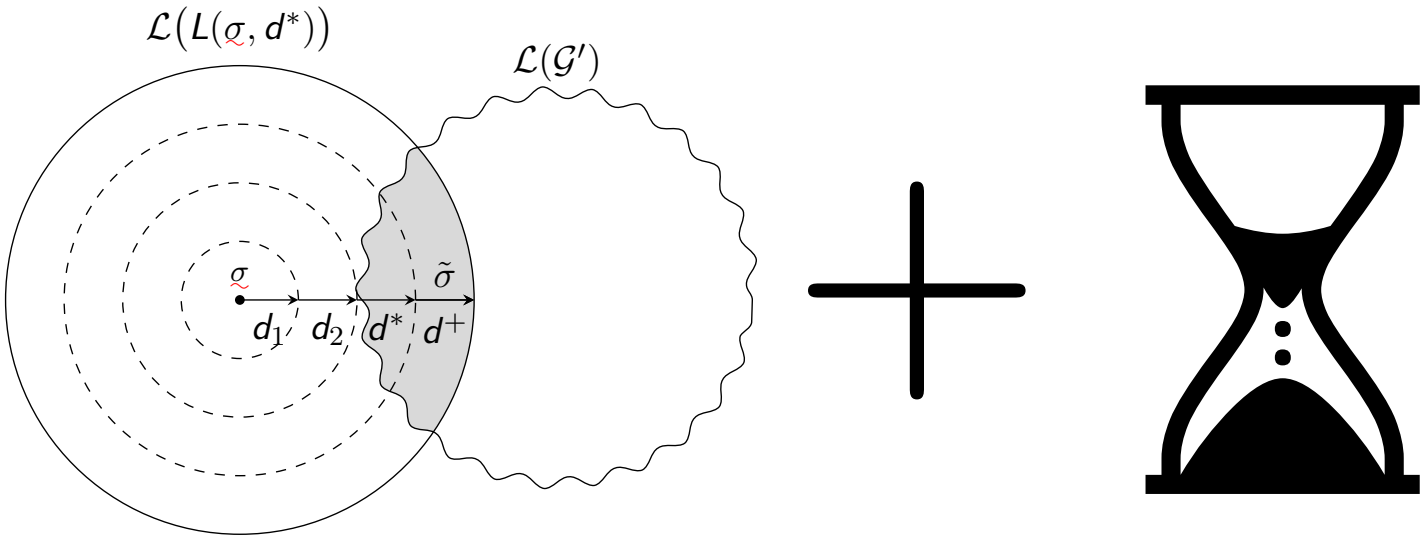
$$R(A, P_\theta) = \underset{\boldsymbol{\sigma} \subseteq A, |\boldsymbol{\sigma}| \leq k}{\operatorname{argmax}} \sum_{\sigma \in \boldsymbol{\sigma}} \mathrm{P}_\theta(\sigma) \tag{1}$$

Problem Statement: Temporal constraints

Find every syntactically admissible edit $\{\sigma' \in \ell \mid \Delta(\underset{\sim}{\sigma}, \sigma') \leq d\}$, ranked by a probability metric P_θ, and return them in a reasonable amount of time.

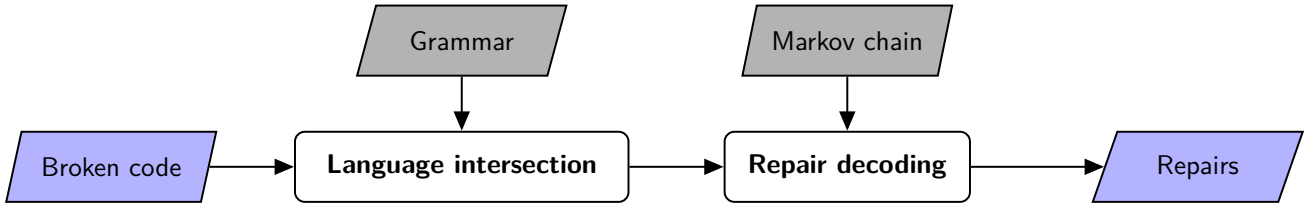
Definition (Linear convergence)

Given a finite CFL, ℓ, we want a generating function, $\varphi : \mathbb{N}_{<|\ell|} \to 2^\ell$, that converges linearly in expectation, i.e., $\mathbb{E}_{i \in [1,n]} |\varphi(i)| \propto n$.

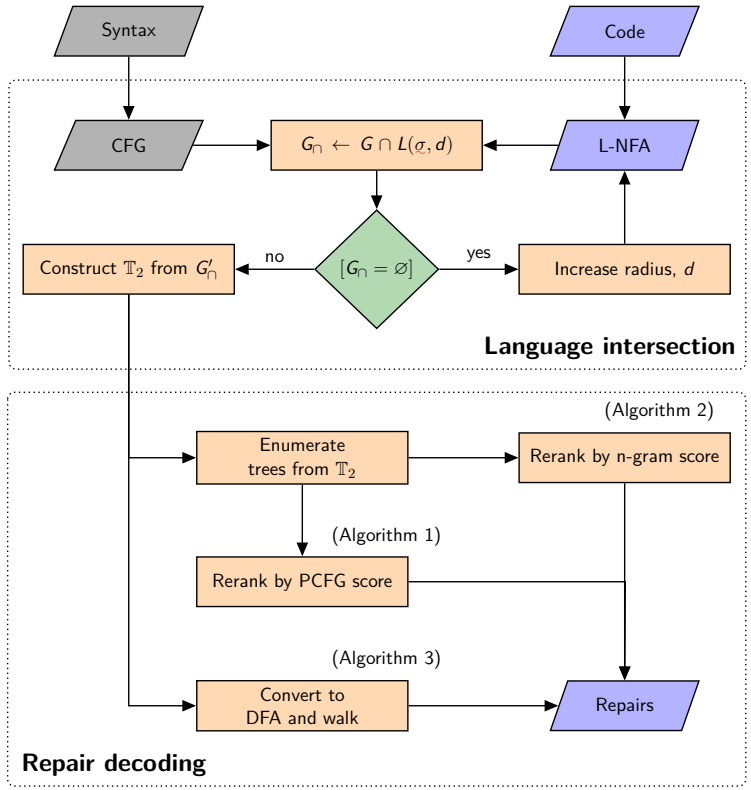


Natural language: *Retrieve as many syntactically valid repairs as possible within a small neighborhood and time frame, ranked by naturalness.*

High-level architecture overview



Our syntax repair procedure can be described in three high-level steps. First, we generate a synthetic grammar ($G_\cap$) representing the intersection between the syntax (G) and Levenshtein ball around the source code ($\Delta(\underset{\sim}{\sigma}, d)$). During repair extraction, we retrieve as many repairs as possible from the intersection grammar via sampling or enumeration. Finally, we rank all repairs discovered by likelihood.



High-level architecture overview

This process can be depicted as series of staged transformations lowering the CFL intersection problem onto a finite automaton. Below, we consider a simplified version based on the language of balanced parentheses.

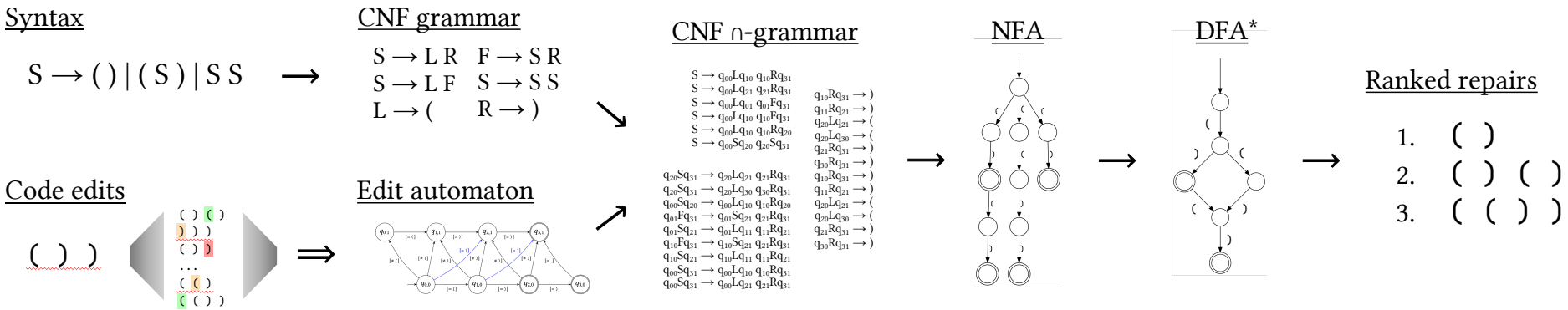
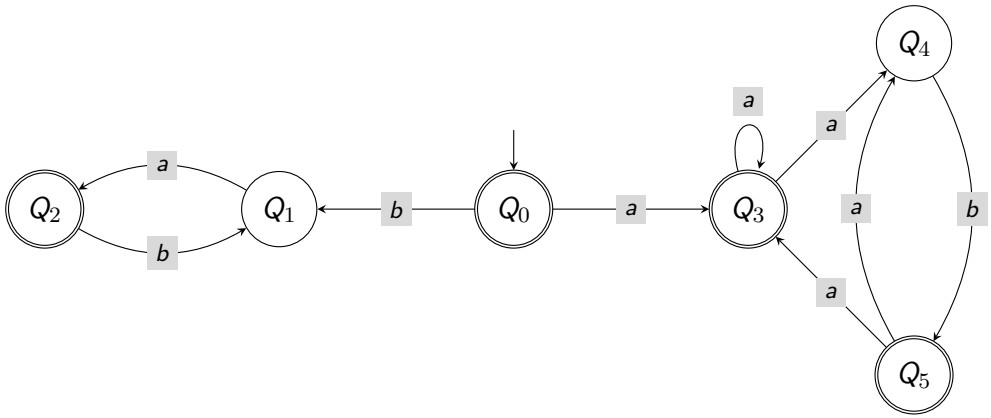


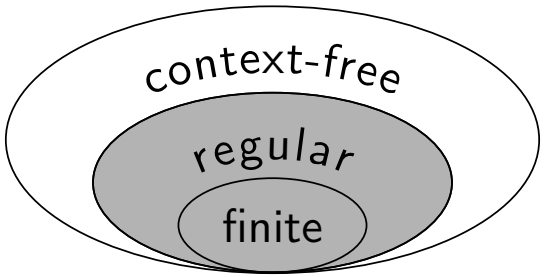
Figure: Simplified dataflow of the language intersection pipeline. Given a grammar and broken code fragment, we (1) create a automaton generating the language of small edits, then (2) construct a grammar representing the intersection of the two languages. This grammar can be (3) converted into a finite automaton, (4) determinized, then (5) decoded to produce a list of repairs.

Background: Regular grammars

A regular grammar (RG) is a quadruple $\mathcal{G} = \langle V, \Sigma, P, S \rangle$ where V are nonterminals, Σ are terminals, $P : V \times (V \cup \Sigma)^{\leq 2}$ are the productions, and $S \in V$ is the start symbol, i.e., all productions are of the form $A \to a$, $A \to aB$ (right-regular), or $A \to Ba$ (left-regular). E.g., the following RG and NFA correspond to the language defined by the *regex* `(a(ab)*)*(ba)*`:

$$
\begin{aligned}
S &\to Q_0 \mid Q_2 \mid Q_3 \mid Q_5 \\
Q_0 &\to \varepsilon \\
Q_1 &\to Q_0 b \mid Q_2 b \\
Q_2 &\to Q_1 a \\
Q_3 &\to Q_0 a \mid Q_3 a \mid Q_5 a \\
Q_4 &\to Q_3 a \mid Q_5 a \\
Q_5 &\to Q_4 b
\end{aligned}
$$





Levenshtein automaton customization

Consider the string $\sigma = (\ )\ )$ and the alphabet $\Sigma = \{), (\}$. Every string within one edit of σ is recognized by an NFA with the following structure:

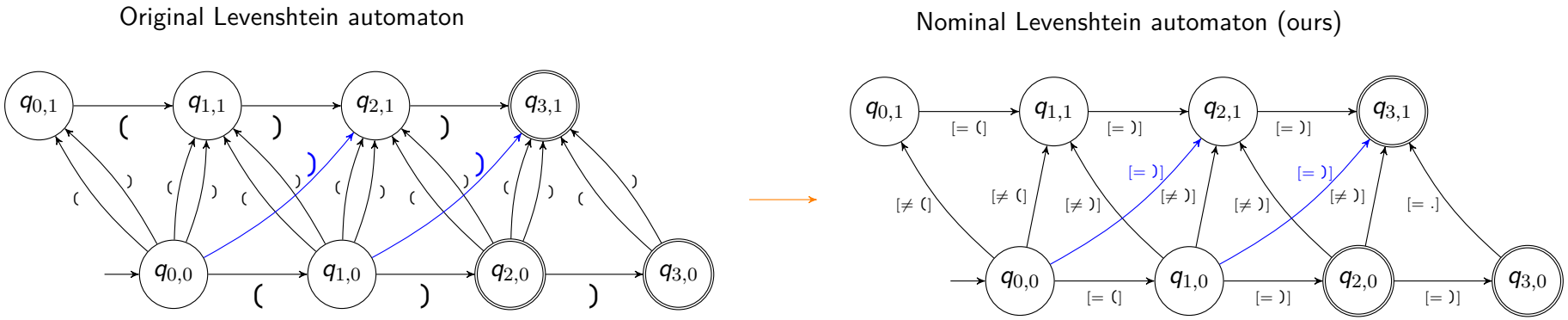


Figure: Automaton recognizing every single patch of the broken string $(\ )\ )$ within Levenshtein distance 1. We nominalize the original Levenshtein automaton, ensuring upward arcs denote a mutation, and replace terminals with a symbolic predicate, which deduplicates parallel arcs in large alphabets.

`https://fulmicoton.com/posts/levenshtein/#observations-lets-count-states`

Levenshtein reachability

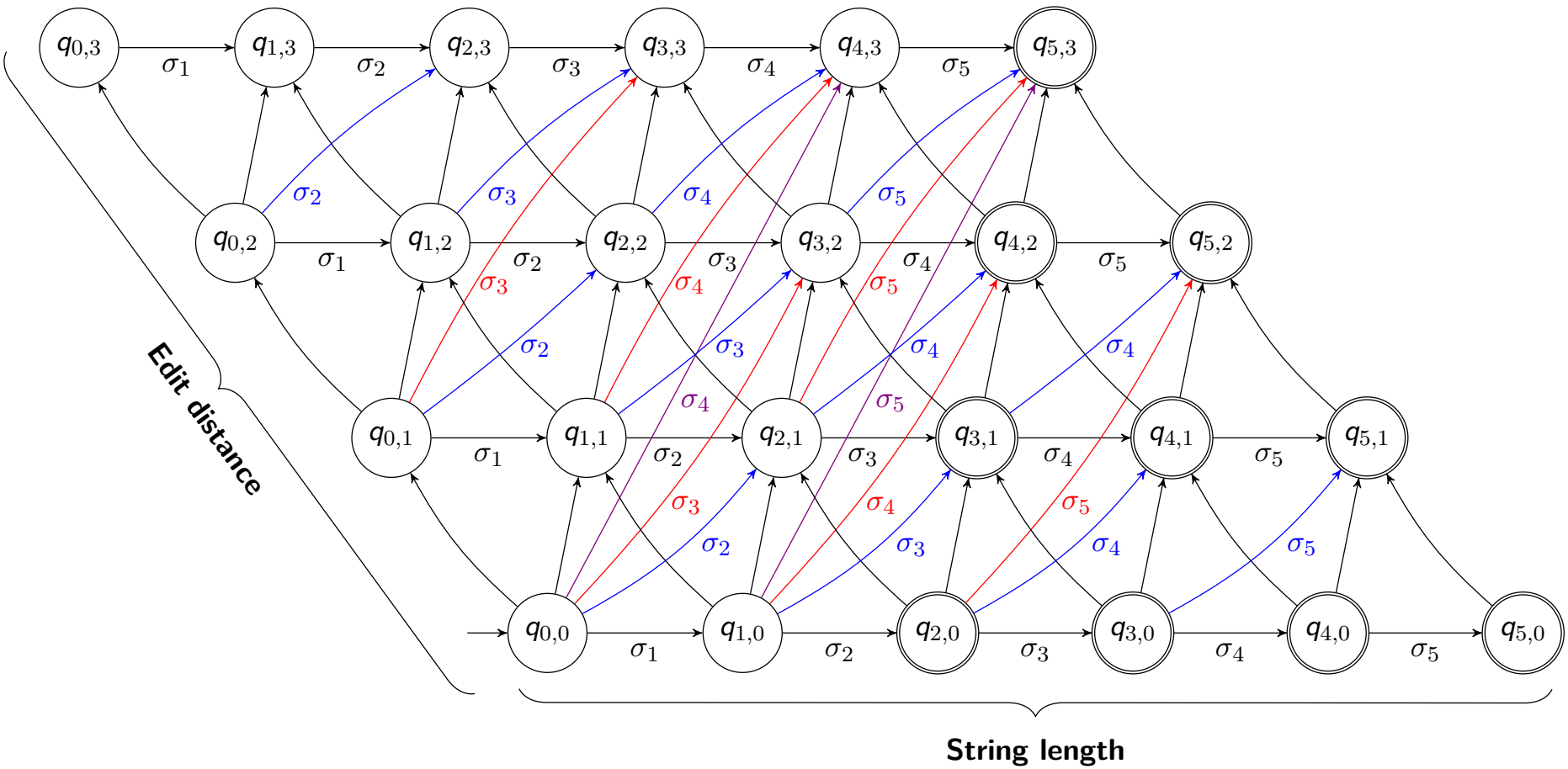
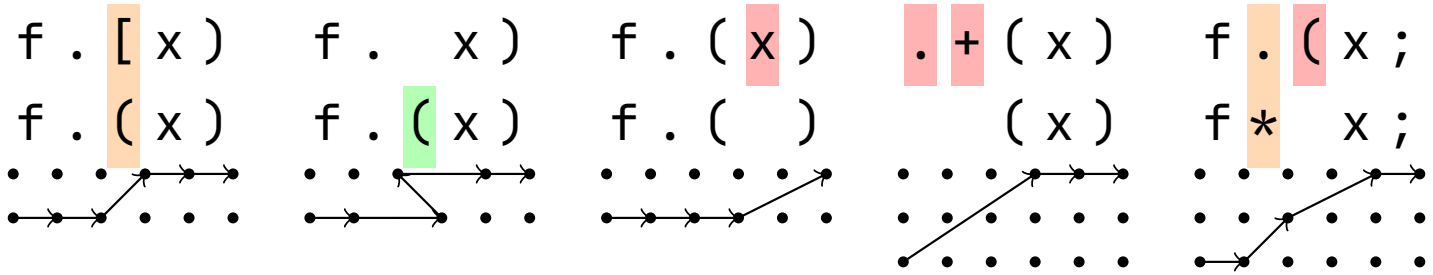


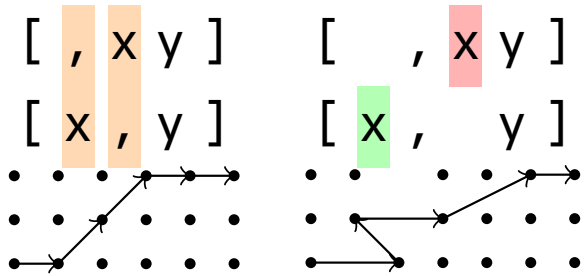
Figure: Bounded Levenshtein reachability from $\sigma : \Sigma^n$ is expressible as an NFA populated by accept states within radius k of $S = q_{n,0}$, which accepts all strings σ' within Levenshtein radius k of σ.

Geometrically interpreting the edit calculus

Each arc plays a specific role. ↕ handles insertions, ↗ handles substitutions and ⋰⋱ handles deletions of ≥ 1 tokens. Consider some illustrative cases:



Note that the same $\langle \underset{\sim}{\sigma}, \sigma' \rangle$ pair can have multiple Levenshtein alignments:



Non-uniqueness of geodesics has implications for CFG ∩ L-NFA ambiguity.

The nominal Levenshtein automaton

The original Levenshtein automaton (Schulz & Stoyan, 2002):

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

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

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

We modify the original automaton with a nominal predicate:

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

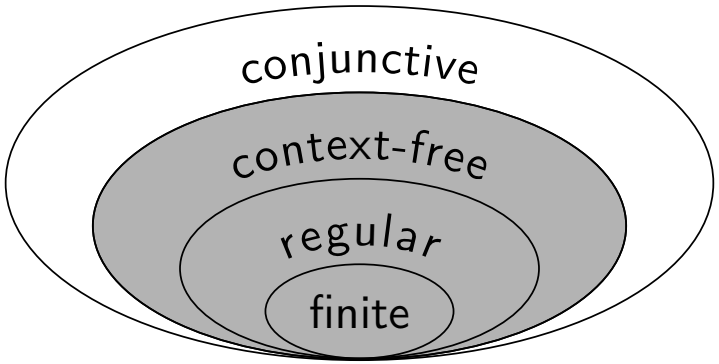
$$\frac{i \in [1, n] \quad j \in [0, k]}{(q_{i-1,j} \xrightarrow{[= \sigma_i]} q_{i,j}) \in \delta} \rightarrow \qquad \frac{d \in [1, d_{\max}] \quad i \in [d + 1, n] \quad j \in [d, k]}{(q_{i-d-1,j-d} \xrightarrow{[= \sigma_i]} q_{i,j}) \in \delta} \nearrow\!\!\rightarrow$$

Background: Context-free grammars

In a context-free grammar $\mathcal{G} = \langle V, \Sigma, P, S \rangle$ all productions are of the form $P : V \times (V \cup \Sigma)^+$, i.e., RHS may contain any number of nonterminals, V. Recognition decidable in $\mathcal{O}(n^\omega)$, n.b. CFLs are **not** closed under $\cap$!

For example, consider the grammar $S \to SS \mid (S) \mid ()$. This represents the language of balanced parentheses, e.g. $(), ()(), (()), ()(()), (()()), (())() \ldots$

Every CFG has a normal form $P^* : V \times (V^2 \mid \Sigma)$, i.e., every production can be refactored into either $v_0 \to v_1 v_2$ or $v_0 \to \sigma$, where $v_{0\ldots2} : V$ and $\sigma : \Sigma$, e.g., $\{S \to SS \mid (S) \mid ()\} \Leftrightarrow^* \{S \to XR \mid SS \mid LR, L \to (, R \to), X \to LS\}$



Background: CFGs as substructural logics

Context-free grammars can also be viewed as a very weak substructural logic where exchange and contraction are disallowed. It has no type constructor and the only inference rule is cut.

This can be enriched by considering categorial grammars, which associates to each symbol a set of types, however the resulting expressivity is (mostly) unchanged, as proven by Pentus in 1991, but Lambek argues the equivalent CFG is typically less comprehensible in practice.

Background: Closure properties of formal languages

Formal languages are not always closed under set-theoretic operations, e.g., CFL $\cap$ CFL is not CFL in general. Let $\cdot$ denote concatenation, $*$ be Kleene star, and $\complement$ be complementation:

	$\cup$	$\cap$	$\cdot$	$*$	$\complement$
Finite[1]	✓	✓	✓	✓	✓
Regular[1]	✓	✓	✓	✓	✓
Context-free[1,2]	✓	✗†	✓	✓	✗
Conjunctive[1,2]	✓	✓	✓	✓	?
Context-sensitive[2]	✓	✓	✓	+	✓
Recursively Enumerable[2]	✓	✓	✓	✓	✗

We would like a language family that is (1) tractable, i.e., has polynomial recognition and search complexity and (2) reasonably expressive, i.e., can represent syntactic properties of real-world programming languages.

† However, CFLs are closed under intersection with regular languages.

The Bar-Hillel construction and its specialization

The original Bar-Hillel construction provides a way to construct a grammar for the intersection of a regular and context-free language.

$$\frac{q \in I \quad r \in F}{(S \to qSr) \in P_\cap}\downarrow \qquad \frac{(A \to a) \in P \qquad (q \xrightarrow{a} r) \in \delta}{(qAr \to a) \in P_\cap}\uparrow$$

$$\frac{(w \to xz) \in P \qquad p, q, r \in Q}{(pwr \to (pxq)(qzr)) \in P_\cap}\bowtie$$

We specialize the Bar-Hillel construction to nominal Levenshtein automata:

$$\frac{(A \to a) \in P \qquad (q \xrightarrow{[\cdot]} r) \in \delta \qquad a[\cdot]}{(qAr \to a) \in P_\cap}\hat{\uparrow}$$

$$\frac{w \lhd pr \quad x \lhd pq \quad z \lhd qr \qquad (w \to xz) \in P \qquad p, q, r \in Q}{(pwr \to (pxq)(qzr)) \in P_\cap}\hat{\bowtie}$$

Where $\lhd$ denotes compatibility between the Parikh map of a nonterminal and Levenshtein margin between two NFA states, see our paper for details.

Parsing for linear algebraists

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

$$s_1 \otimes s_2 := \{C \mid \langle A, B \rangle \in s_1 \times s_2, (C \to AB) \in P\}$$

e.g., $\{\text{A} \to \text{BC}, \text{C} \to \text{AD}, \text{D} \to \text{BA}\} \subseteq \text{P} \vdash \{\text{A, B, C}\} \otimes \{\text{B, C, D}\} = \{\text{A, C}\}$

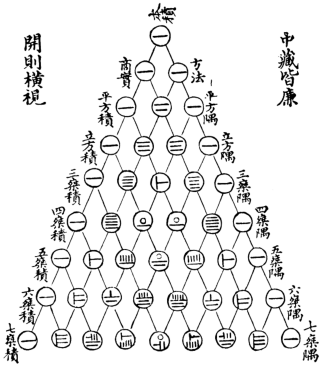
If we define $\sigma_r^{\pitchfork} := \{w \mid (w \to \sigma_r) \in P\}$, then initialize $M^0_{r+1=c}(\mathcal{G}', e) := \sigma_r^{\pitchfork}$ and solve for the fixpoint $M^* = M + M^2$,

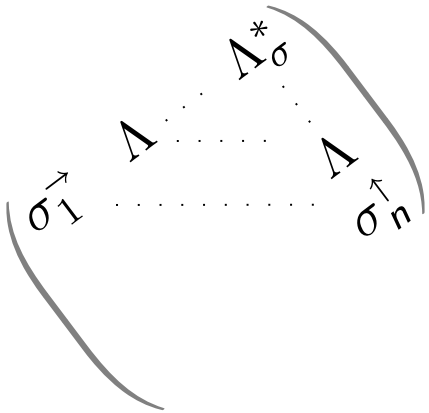
$$M^0 := \begin{pmatrix} \varnothing & \sigma_1^{\rightarrow} & \varnothing & \cdots & \varnothing \\ \vdots & \ddots & & & \vdots \\ & & & & \varnothing \\ & & & & \sigma_n^{\uparrow} \\ \varnothing & \cdots & & & \varnothing \end{pmatrix} \Rightarrow \ldots \Rightarrow M^* = \begin{pmatrix} \varnothing & \sigma_1^{\rightarrow} & \Lambda & \cdots & \Lambda_\sigma^* \\ \vdots & \ddots & & & \vdots \\ & & & & \Lambda \\ & & & & \sigma_n^{\uparrow} \\ \varnothing & \cdots & & & \varnothing \end{pmatrix}$$

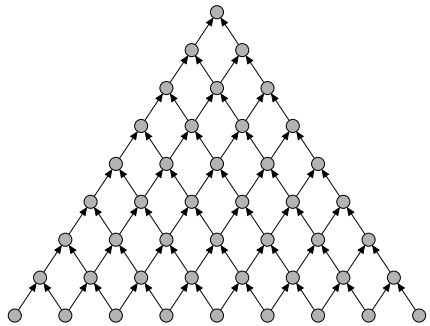
$S \Rightarrow^* \sigma \iff \sigma \in \mathcal{L}(\mathcal{G})$ iff $S \in \Lambda_\sigma^*$, i.e., $\mathbb{1}_{\Lambda_\sigma^*}(S) \iff \mathbb{1}_{\mathcal{L}(\mathcal{G})}(\sigma)$.

Lattices, Matrices and Trellises

The art of treillage has been practiced from ancient through modern times.







Jia Xian Triangle
Jia, ∼1030 A.D.

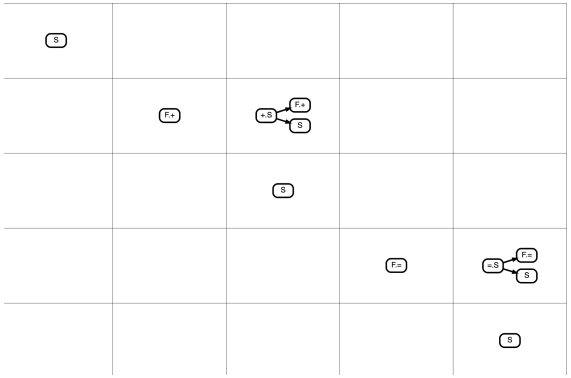
CYK Parsing
Sakai, 1961 A.D.

Trellis Automaton
Dyer, 1980 A.D.

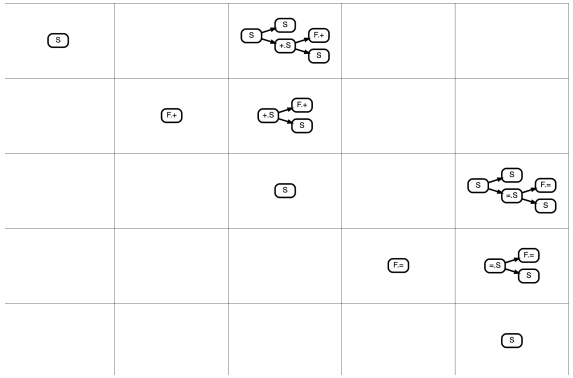
A few observations on algebraic parsing

- The matrix $\mathbf{M}^*$ is strictly upper triangular, i.e., nilpotent of degree n
- Recognizer can be translated into a parser by storing backpointers

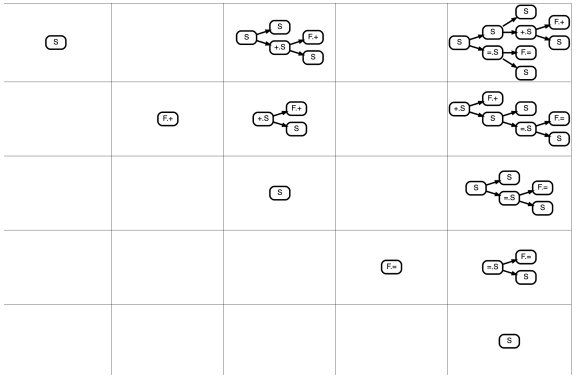
$\mathbf{M}_1 = \mathbf{M}_0 + \mathbf{M}_0^2$



$\mathbf{M}_2 = \mathbf{M}_1 + \mathbf{M}_1^2$



$\mathbf{M}_3 = \mathbf{M}_2 + \mathbf{M}_2^2 = \mathbf{M}_4$



- The $\otimes$ operator is *not* associative: $S \otimes (S \otimes S) \neq (S \otimes S) \otimes S$
- Built-in error recovery: nonempty submatrices = parsable fragments
- `seekFixpoint { it + it * it }` is sufficient but unnecessary
- If we had a way to solve for $\mathbf{M} = \mathbf{M} + \mathbf{M}^2$ directly, power iteration would be unnecessary, could solve for $\mathbf{M} = \mathbf{M}^2$ above superdiagonal

Satisfiability + holes

- Can be lowered onto a Boolean tensor $\mathbb{B}_2^{n \times n \times |V|}$ (Valiant, 1975)
- Binarized CYK parser can be efficiently compiled to a SAT solver
- Enables sketch-based synthesis in either σ or $\mathcal{G}$: just use variables!
- We simply encode the characteristic function, i.e. $\mathbb{1}_{\subseteq V} : V \to \mathbb{Z}_2^{|V|}$
- $\oplus, \otimes$ are defined as $\boxplus, \boxtimes$, so that the following diagram commutes:

$$\begin{array}{ccc} 2^V \times 2^V & \xrightarrow{\oplus/\otimes} & 2^V \\ {\scriptstyle \mathbb{1}^{-2}}\big\updownarrow{\scriptstyle \mathbb{1}^2} & & {\scriptstyle \mathbb{1}^{-1}}\big\updownarrow{\scriptstyle \mathbb{1}} \\ \mathbb{Z}_2^{|V|} \times \mathbb{Z}_2^{|V|} & \xrightarrow[\boxplus/\boxtimes]{} & \mathbb{Z}_2^{|V|} \end{array}$$

- These operators can be lifted into matrices/tensors in the usual way
- In most cases, only a few nonterminals are active at any given time

Satisfiability + holes

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

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

Semiring algebras: Part I

The prior solution tell us whether $A(\sigma)$ is nonempty, but forgets the solution(s). To solve for $A(\sigma)$, a naïve approach accumulates a mapping of nonterminals to sets of strings. Letting $D = V \to \mathcal{P}(\Sigma^*)$, we define $\oplus, \otimes : D \times D \to D$. Initially, we construct $M_0[r+1 = c] = p(\sigma_r)$ using:

$$p(s : \Sigma) \mapsto \{w \mid (w \to s) \in P\} \text{ and } p(\_) \mapsto \bigcup_{s \in \Sigma} p(s)$$

$p(\cdot)$ constructs the superdiagonal, then we solve for Λ^*_σ using the algebra:

$$X \oplus Z \mapsto \left\{ w \overset{+}{\Rightarrow} (X \circ w) \cup (Z \circ w) \mid w \in \pi_1(X \cup Z) \right\}$$

$$X \otimes Z \mapsto \bigoplus_{w,x,z} \left\{ w \overset{+}{\Rightarrow} (X \circ x)(Z \circ z) \mid (w \to xz) \in P, x \in X, z \in Z \right\}$$

After M_∞ is attained, the solutions can be read off via $\Lambda^*_\sigma \circ S$. The issue here is exponential growth when eagerly computing the transitive closure.

Semiring algebras: Part II

The prior encoding can be improved using an ADT $\mathbb{T}_3 = (V \cup \Sigma) \rightharpoonup \mathbb{T}_2$ where $\mathbb{T}_2 = (V \cup \Sigma) \times (\mathbb{N} \rightharpoonup \mathbb{T}_2 \times \mathbb{T}_2)$. We construct $\hat{\sigma}_r = \dot{p}(\sigma_r)$ using:

$$\dot{p}(s : \Sigma) \mapsto \left\{ \mathbb{T}_2\big(w, \big[\langle \mathbb{T}_2(s), \mathbb{T}_2(\varepsilon)\rangle\big]\big) \mid (w \to s) \in P \right\} \text{ and } \dot{p}(\_) \mapsto \bigoplus_{s \in \Sigma} p(s)$$

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

$$X \oplus Z \mapsto \bigcup_{k \in \pi_1(X \cup Z)} \left\{ k \Rightarrow \mathbb{T}_2(k, x \cup z) \mid x \in \pi_2(X \circ k), z \in \pi_2(Z \circ k) \right\}$$

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

Semiring algebras: Part III

$$L(p) = 1 + pL(p) \qquad P(a) = \Sigma + V + VL\left(V^2 P(a)^2\right)$$

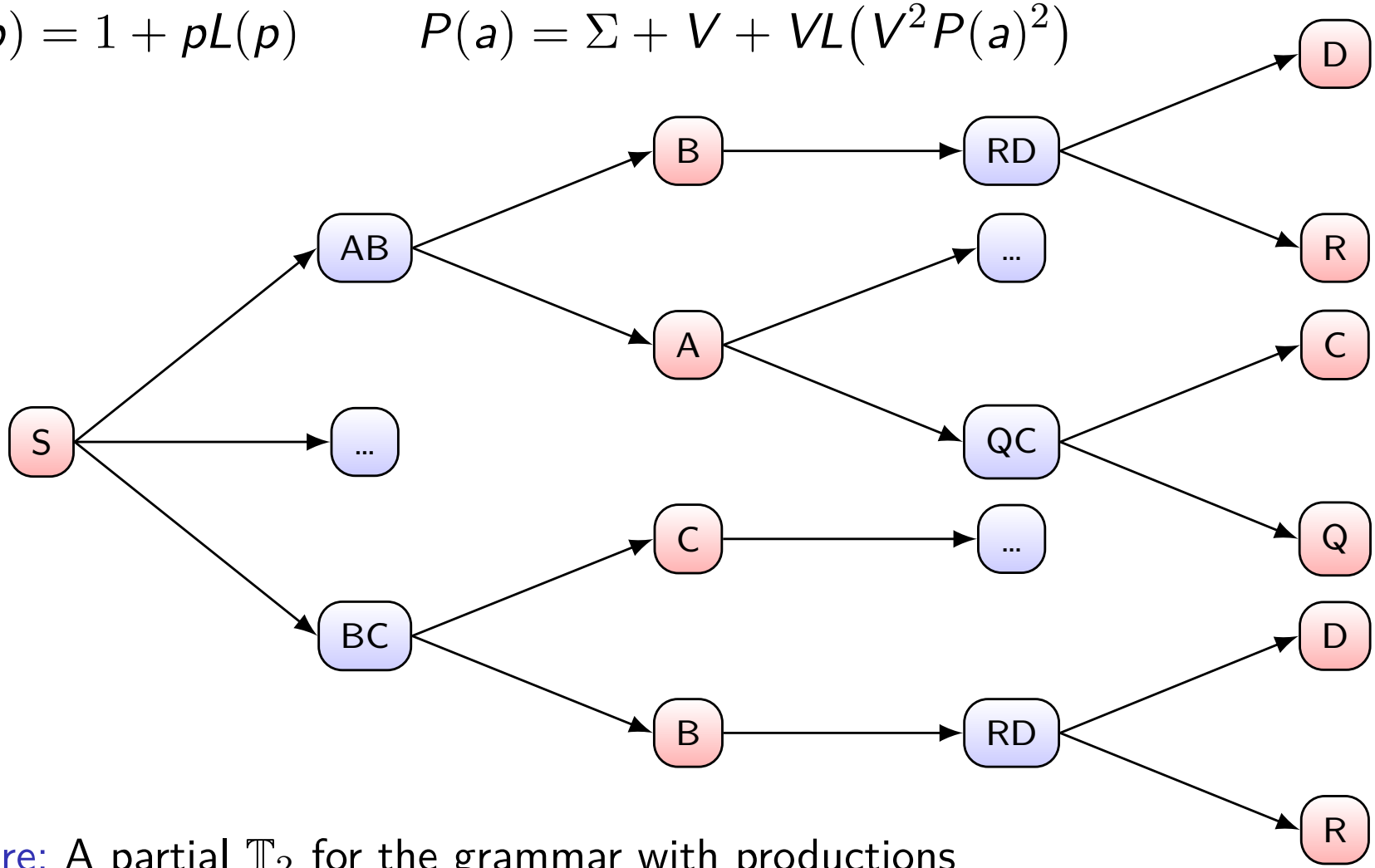


Figure: A partial $\mathbb{T}_2$ for the grammar with productions $P = \{S \to BC \mid \ldots \mid AB, B \to RD \mid \ldots, A \to QC \mid \ldots\}$.

Sampling trees with replacement

Given a probabilistic CFG whose productions indexed by each nonterminal are decorated with a probability vector $\mathbf{p}$ (this may be uniform in the non-probabilistic case), we define a tree sampler $\Gamma : \mathbb{T}_2 \rightsquigarrow \mathbb{T}$ which recursively samples children according to a Multinoulli distribution:

$$\Gamma(T) \mapsto \begin{cases} \text{Multi}\big(\texttt{children}(T), \mathbf{p}\big) & \text{if } T \text{ is a } \texttt{root} \\ \big\langle \Gamma\big(\pi_1(T)\big), \Gamma\big(\pi_2(T)\big) \big\rangle & \text{if } T \text{ is a } \texttt{child} \end{cases}$$

This is closely related to the generating function for the ordinary Boltzmann sampler from analytic combinatorics,

$$\Gamma C(x) \mapsto \begin{cases} \text{Bern}\left(\frac{A(x)}{A(x)+B(x)}\right) \to \Gamma A(x) \mid \Gamma B(x) & \text{if } \mathcal{C} = \mathcal{A} + \mathcal{B} \\ \big\langle \Gamma A(x), \Gamma B(x) \big\rangle & \text{if } \mathcal{C} = \mathcal{A} \times \mathcal{B} \end{cases}$$

however unlike Duchon et al. (2004), rejection is unnecessary to ensure exact-size sampling, as all trees in $\mathbb{T}_2$ will necessarily be the same size.

A pairing function for replacement-free tree sampling

The total number of trees induced by a given sketch template is given by:

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

To sample from $\mathbb{T}_2$ without replacement, we define a pairing function:

$$\varphi(T : \mathbb{T}_2, i : \mathbb{Z}_{|T|}) \mapsto \begin{cases} \texttt{BTree}\big(\texttt{root}(T)\big) & \text{if } T \text{ is a leaf,} \\ \textbf{let } r = |\texttt{children}(T)|, & \\ \quad F(n) = \sum_{\langle l, r \rangle \in \texttt{children}[0 \ldots n]} |l| \cdot |r|, & \\ \quad F^{-1}(u) = \inf\big\{x \mid u \leq F(x)\big\}, & \\ \quad q = i - F\big(F^{-1}(i)\big), & \\ \quad l, r = \texttt{children}[t], & \\ \quad q_1, q_2 = \big\langle \lfloor \frac{q}{|r|} \rfloor, q \pmod{|r|} \big\rangle, & \\ \quad T_1, T_2 = \big\langle \varphi(l, q_1), \varphi(r, q_2) \big\rangle \textbf{ in} & \\ \texttt{BTree}\big(\texttt{root}(T), T_1, T_2\big) & \text{otherwise.} \end{cases}$$

Enumerative search with reranking

Given $\sigma : \Sigma^*$, let $P_\theta(\sigma) = \prod_{i=1}^{|\sigma|} P_\theta(\sigma_i \mid \sigma_{i-1} \dots \sigma_{i-n})$. This defines an ordering over Σ^*. Then, for each retrieved set $\sigma \in \hat{a} \subseteq A$ drawn before timeout, we score the repair and return $\hat{A}$ in ascending order:

Algorithm Enumerative tree sampling with n-gram reranking

Require: $T : \mathbb{T}_2$ intersection grammar, $P_\theta : \Sigma^d \to \mathbb{R}$ Markov chain

1: $\hat{A} \leftarrow \varnothing, \mathsf{seed} \leftarrow 0$ ▷ Initialize set of repairs.
2: **for** $\mathsf{seed} < |T|$ and uninterrupted **do**
3: $\quad t \leftarrow \varphi'(T, \mathsf{seed{+}{+}})$ ▷ Decode fresh tree and add.
4: $\quad \hat{A} \leftarrow \hat{A} \cup \{\mathscr{Y}(t)\}$
5: **return** $[\sigma \in \hat{A}$ **ranked by** $\mathrm{NLL}(\sigma)]$ ▷ Rerank by n-gram likelihood.

The issue is, the bijection is defined over labeled binary trees and does not guarantee unique strings, as the CFG may be ambiguous. In practice, this matters if we care about $\mathcal{O}(n)$ convergence!

Potential ambiguity of Levenshtein-Bar-Hillel grammars

The previous technique enumerates parse trees in a given $\mathbb{T}_2$, but does not guarantee string uniqueness since the CFG may be ambiguous.

Lemma

If the FSA, α, is ambiguous, the intersection CFG, $G_\cap$, 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 can eliminate ambiguity and thereby improve the rate of convergence for natural syntax repair by first translating $\mathbb{T}_2$ into an FSA.

Existence of an FSA that generates $\mathcal{L}(G_\cap)$

There is an FSA generating $\mathcal{L}(G_\cap)$. We first show this non-constructively:

Lemma

The intersection grammar, $G_\cap$, is acyclic.

Proof.

Assume $G_\cap$ is cyclic. Then $\mathcal{L}(G_\cap)$ must be infinite. But since $G_\cap$ generates $\ell \cap \mathcal{L}(\alpha)$ by construction and α is acyclic, $\mathcal{L}(G_\cap)$ is necessarily finite. Therefore, $G_\cap$ must not be cyclic. □

Since $G_\cap$ is acyclic and thus finite, it must be representable as an FSA. Using an FSA for decoding has many advantages, notably, it can be efficiently minimized and decoded in order of n-gram likelihood using a Markov chain or standard pretrained autoregressive language model.

Translating from T_2 to a DFA

Let $+, * : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ be automata operators satisfying the property $\mathcal{L}(A_1 + A_2) = \mathcal{L}(A_1) \cup \mathcal{L}(A_2)$, and $\mathcal{L}(A_1 * A_2) = \mathcal{L}(A_1) \times \mathcal{L}(A_2)$. We can translate $\mathbb{T}_2$ to $\mathcal{A}$, as follows, recalling FSAs are closed over $+, *$:

$$\mathcal{Y}(T : \mathbb{T}_2) \mapsto \begin{cases} \alpha \mid \mathcal{L}(\alpha) = \{T\} & T : \Sigma, \\ \sum_{\langle T_1, T_2 \rangle \in \texttt{children}(T)} \mathcal{Y}(T_1) * \mathcal{Y}(T_2) & T : VL\left(V^2 P(a)^2\right) \end{cases}$$

In the case of LBH intersections, $\mathcal{Y}(G'_{\cap})$ yields $\alpha : \mathcal{A} \mid \mathcal{L}(\alpha) = \ell \cap L(\underset{\sim}{\sigma}, d)$, which can be minimized via Brzozowski's algorithm then decoded:

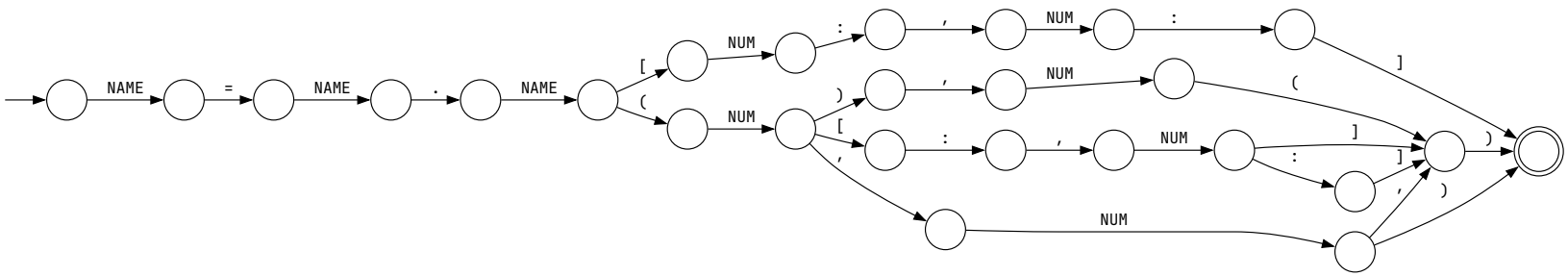


Figure: $L(\texttt{NAME = NAME . NAME (NUM : , NUM :)}, 2) \cap \ell_{\text{PYTHON}}$

Decoding the DFA in order of normalized log likelihood

Algorithm Steerable DFA walk

Require: $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$ DFA, $P_\theta : \Sigma^d \to \mathbb{R}$ Markov chain

$\mathcal{T} \leftarrow \varnothing$ total trajectories, $\mathcal{P} \leftarrow \left[\langle \varepsilon, i, 0 \rangle \mid i \in I\right]$ partial trajectories
repeat
  let $\langle \sigma, q, \gamma \rangle = \mathsf{head}(\mathcal{P})$ **in**
    $\mathbf{T} = \left\{\langle s\sigma, q', \gamma - \log P_\theta(s \mid \sigma_{1..d-1}) \rangle \mid (q \xrightarrow{s} q') \in \delta\right\}$
  for $\langle \sigma, q, \gamma \rangle = T \in \mathbf{T}$ **do**
    if $\exists s : \Sigma, q' : Q \mid (q \xrightarrow{s} q') \in \delta$ **then**
      $\mathcal{P} \leftarrow \mathtt{tail}(\mathcal{P}) \oplus T$ ▷ Add partial trajectory to PQ.
    if $q \in F$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} \oplus T$ ▷ Accepting state reached, add to queue.
until interrupted or $\mathcal{P} = \varnothing$.
return $\left[\sigma_{|\sigma|..1} \mid \langle \sigma, q, \gamma \rangle = T \in \mathcal{T}\right]$ ▷ Return in sorted order

Characteristics of the repair dataset

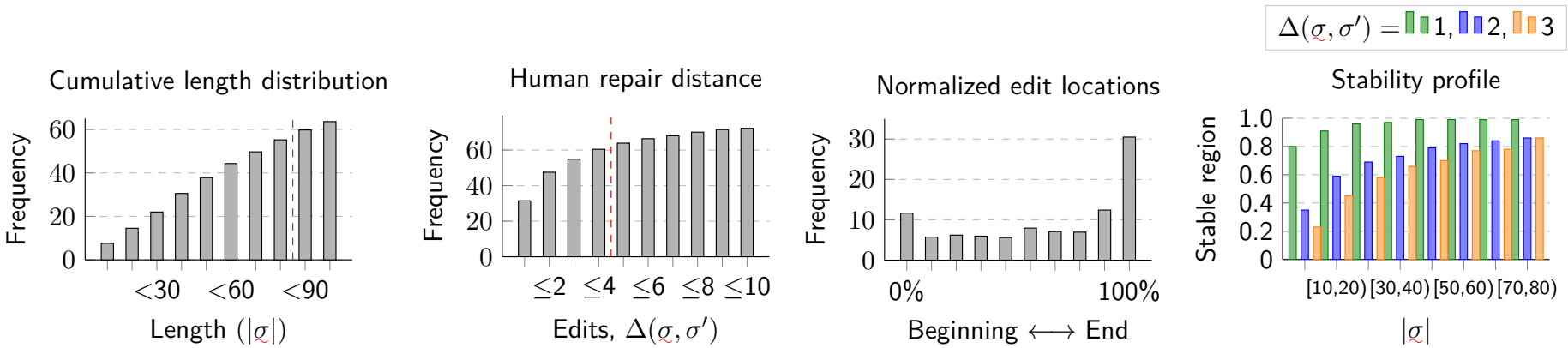


Figure: Repair statistics across the StackOverflow dataset, of which Tidyparse can handle about half in under ∼30s and ∼150 GB. Larger repairs and edit distances are possible, albeit requiring additional time and memory. The stability profile measures the average fraction of all edit locations that were never altered by any repair in the $L\big(\sigma, \Delta(\sigma, \sigma')\big)$-ball across repairs of varying length and distance.

Ranked repair

We train on lexical n-grams using the standard MLE for Markov chains.
To score the repairs, we use the conventional length-normalized NLL:

$$\text{NLL}(\sigma) = -\frac{1}{|\sigma|} \sum_{i=1}^{|\sigma|} \log P_\theta(\sigma_i \mid \sigma_{i-1} \dots \sigma_{i-n}) \tag{2}$$

For each retrieved set $\hat{A} \subseteq A$ drawn before a predetermined timeout and each $\sigma \in \hat{A}$, we score the repair and return $\hat{A}$ in ascending order.
To evaluate the quality of our ranking, we use the Precision@k statistic. Specifically, given a repair model, $R : \Sigma^* \to 2^{\Sigma^*}$ and a parallel corpus, $\mathcal{D}_{\text{test}}$, of errors ($\sigma^\dagger$) and repairs ($\sigma'$), we define Precision@k as:

$$\text{Precision@k}(R) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\langle \sigma^\dagger, \sigma' \rangle \in \mathcal{D}_{\text{test}}} \mathbb{1}\left[\sigma' \in \underset{\boldsymbol{\sigma} \subset R(\sigma^\dagger), |\boldsymbol{\sigma}| \leq k}{\operatorname{argmax}} \sum_{\sigma \in \boldsymbol{\sigma}} \text{NLL}(\sigma)\right] \tag{3}$$

Precision and latency comparison

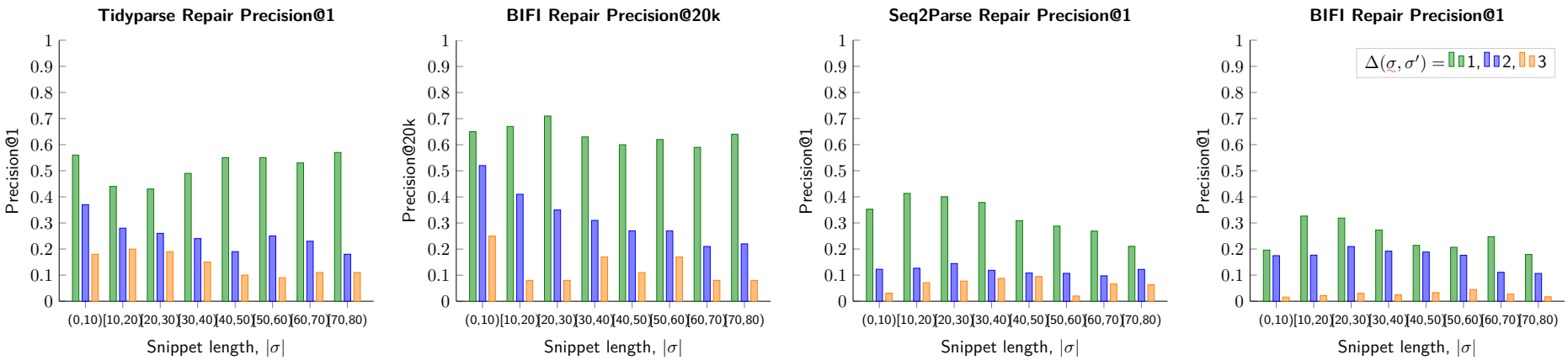


Figure: Tidyparse, Seq2Parse and BIFI repair precision across length and edits.

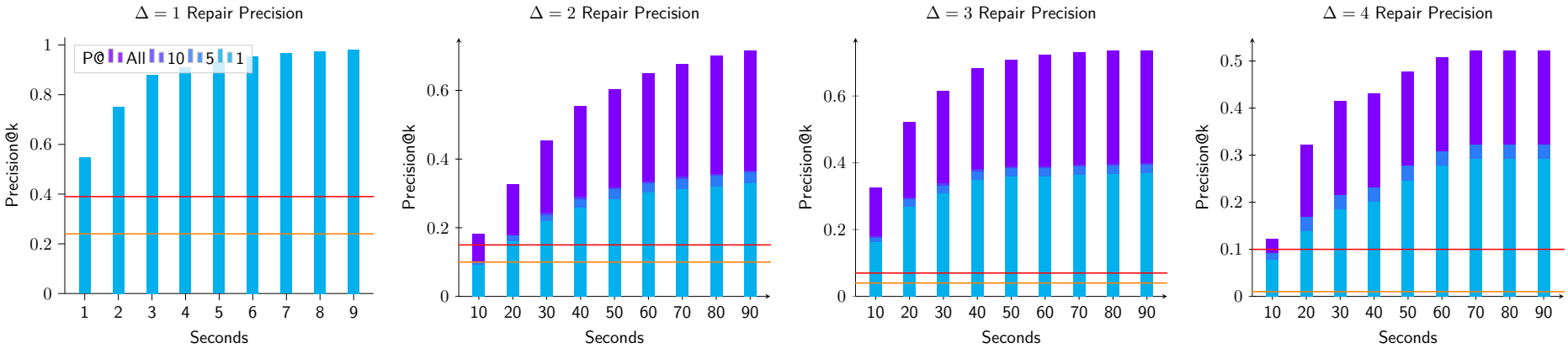
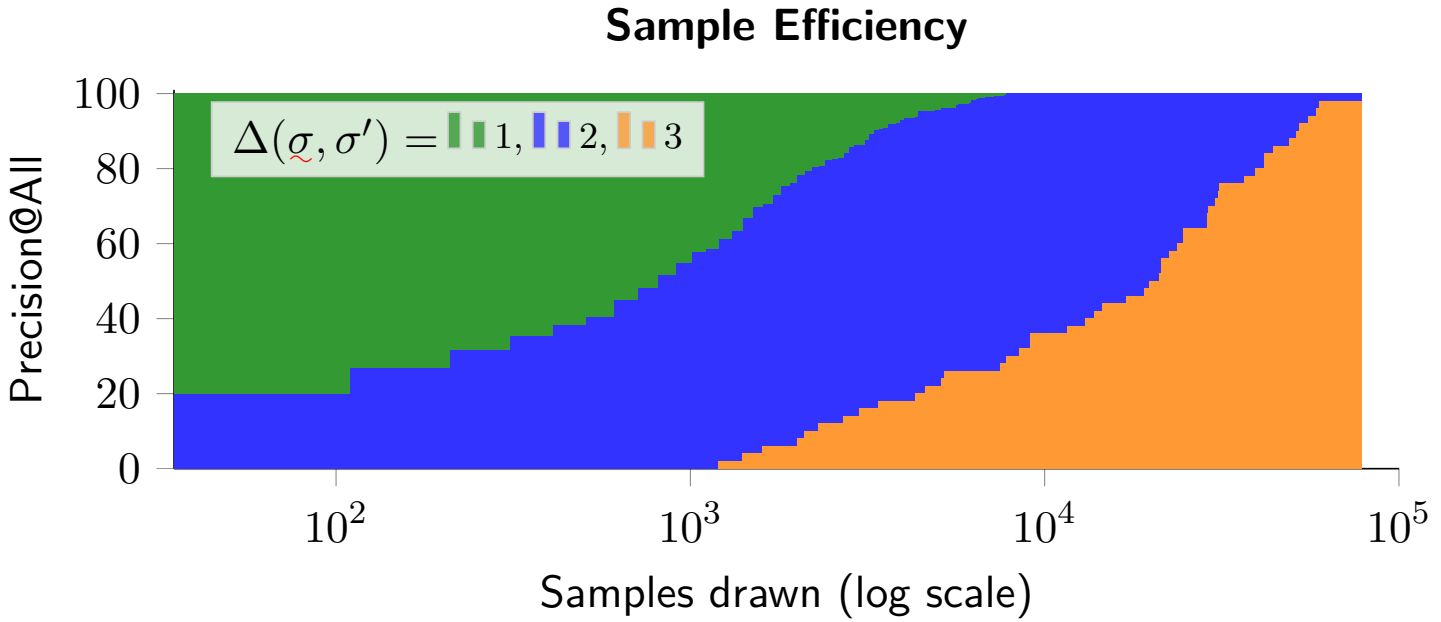
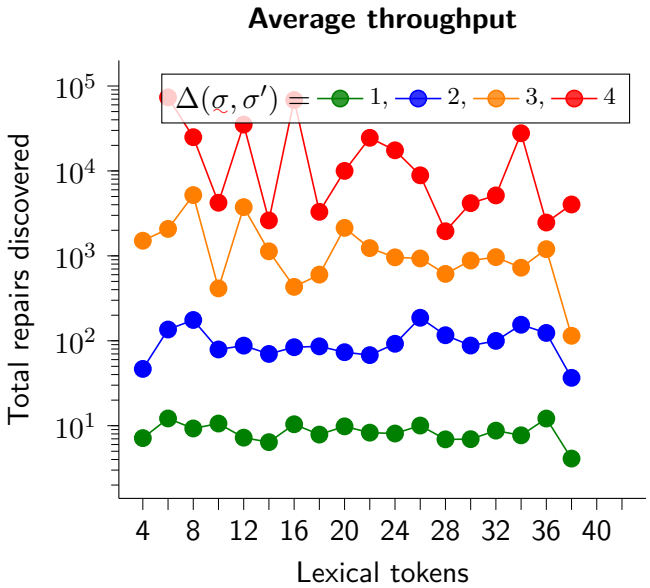
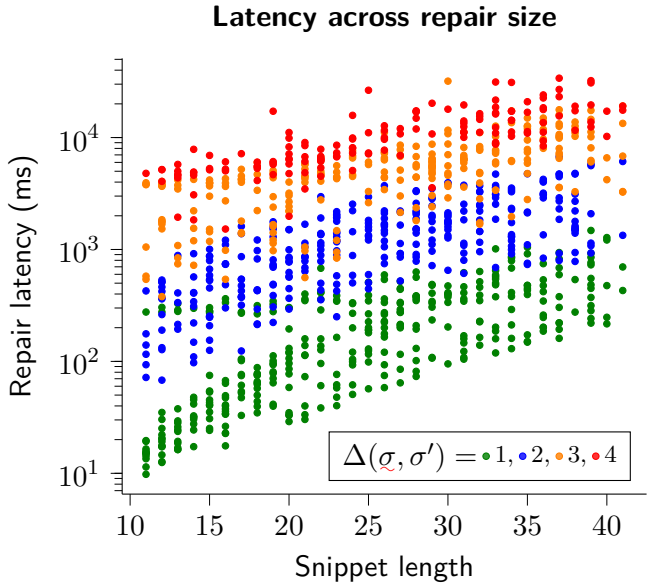


Figure: Latency benchmarks. Note the varying y-axis ranges. The red line marks Seq2Parse and the orange line marks BIFI's Precision@1 on the same repairs.

Results from sample efficiency experiments







Outcomes in the syntax repair pipeline

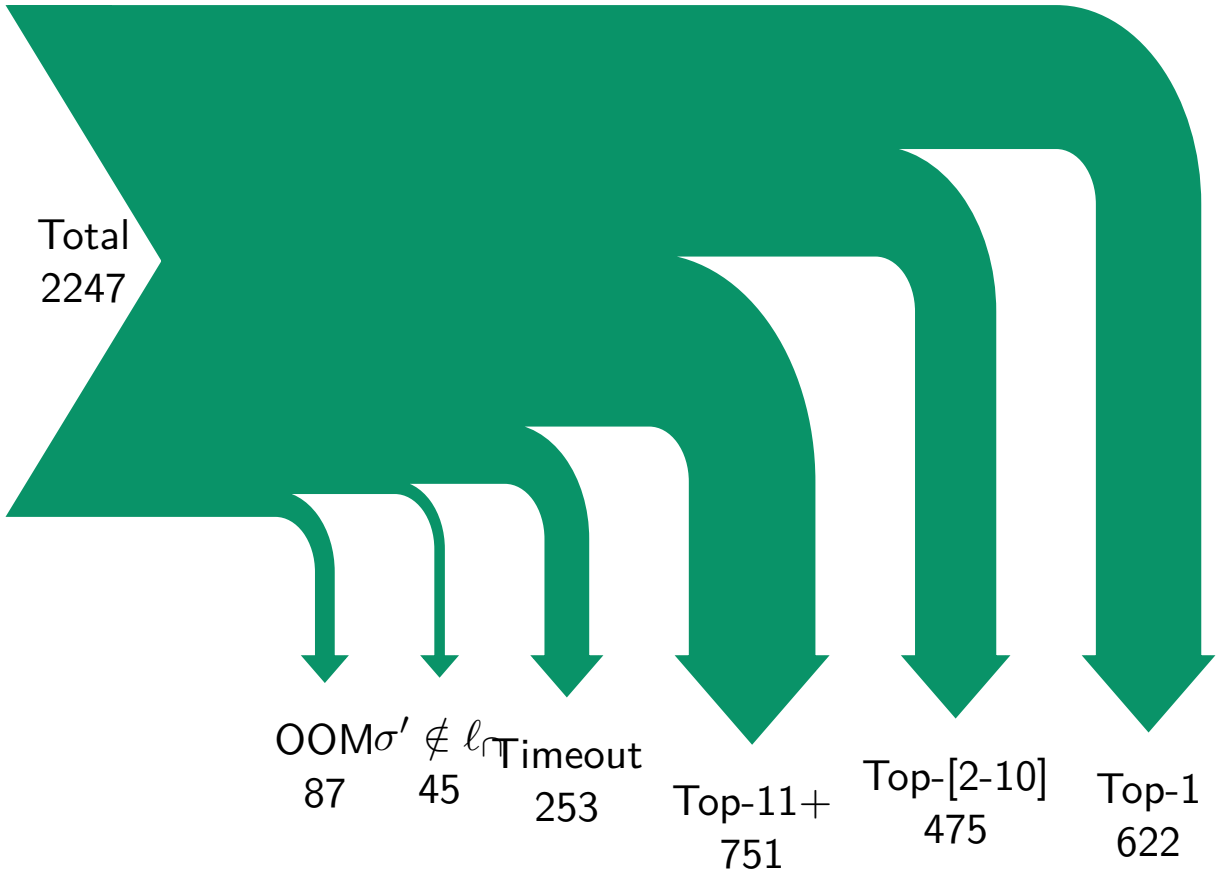


Figure: Sankey diagram of 967 total repair instances sampled uniformly from the StackOverflow Python dataset balanced acrost snippet lengths and edit distances ($\lfloor |\underset{\sim}{\sigma}|/10 \rfloor \in [0, 8], \Delta(\underset{\sim}{\sigma}, \sigma') < 4$) with a sampling timeout of 30s per repair.

Feature comparison matrix

	Sound	**Complete**	**Natural**	**Theory**	**||**	**Tool**
Tidyparse	✓	✓	✓	$\text{CFG}_\cap$	✓	IDE-ready
Seq2Parse	✓†	✗	✓	CFG	✗	Python
BIFI	✗	✗	✓	Σ^*	✗	Python
OrdinalFix	✓	✗	✗	CFG+	✗	Rust
Outlines[1]	✓†	✓†	✓	EBNF	✗	Python
SynCode[1]	✓	✓	✓	EBNF	✗	Python
Aho/Irons	✓	✗	✗	CFG	✗	None

Sound = generated repairs always syntactically valid.
Complete = all valid repairs are eventually generated.
Natural $\approx$ statistically likely / designed to model human preferability.
|| = Trivially parallelizable, i.e., designed to be executed on multiple cores.

[1] Not specifically intended for syntax repair, but can be adapted.
† Claimed by the authors, but counterexamples known to exist.

Abbreviated history of algebraic parsing

- Chomsky & Schützenberger (1959) - The algebraic theory of CFLs
- Cocke–Younger–Kasami (1961) - Bottom-up matrix-based parsing
- Brzozowski (1964) - Derivatives of regular expressions
- Valiant (1975) - first realizes the Boolean matrix correspondence
  - Naïvely, has complexity $\mathcal{O}(n^4)$, can be reduced to $\mathcal{O}(n^\omega)$, $\omega < 2.763$
- Lee (1997) - Fast CFG Parsing $\Longleftrightarrow$ Fast BMM, formalizes reduction
- Might et al. (2011) - Parsing with derivatives (Brzozowski $\Rightarrow$ CFL)
- Bakinova, Okhotin et al. (2010) - Formal languages over GF(2)
- Bernady & Jansson (2015) - Certifies Valiant (1975) in Agda
- Cohen & Gildea (2016) - Generalizes Valiant (1975) to parse and recognize mildly context sensitive languages, e.g. LCFRS, TAG, CCG
- **Considine, Guo & Si (2022) - SAT + Valiant (1975) + holes**
- **Considine, Guo & Si (2024) - Levenshtein Bar-Hillel repairs**

Special thanks

Jin Guo, Xujie Si, David Bieber,
David Chiang, Brigitte Pientka, David Hui,
Ori Roth, Younesse Kaddar, Michael Schröder
Will Chrichton, Kristopher Micinski, Alex Lew
Matthijs Vákár, Michael Coblenz, Maddy Bowers

McGill UNIVERSITY

Mila

Learn more at:

https://tidyparse.github.io