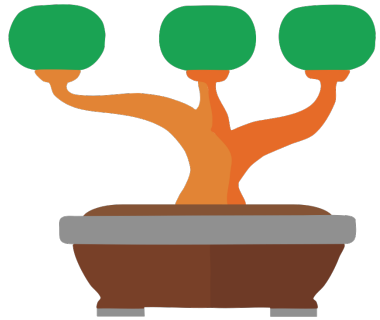
A Pragmatic Approach to Syntax Repair

Breandan Considine, Jin Guo, Xujie Si

McGill University, Mila IQIA

bre@ndan.co

October 26, 2023



Can you spot the error?

Original code	**Human repair**

`form sympy import *`
`x = Symbol('x', real=True)`
`x, re(x), im(x)`	

Can you spot the error?

Original code	Human repair

`form sympy import *`
`x = Symbol('x', real=True)`
`x, re(x), im(x)`
	`from sympy import *`
	`x = Symbol('x', real=True)`
	`x, re(x), im(x)`

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**

`my_list = []`
`for i in range(10);`
`  my_list.append(2*i)`	

Can you spot the error?

Original code	**Human repair**

`my_list = []`
`for i in range(10);`
`  my_list.append(2*i)`
	`my_list = []`
	`for i in range(10):`
	`  my_list.append(2*i)`

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()))`

Can you spot the error?

Original code	Human repair

`try:`
`  something()`
`catch AttributeError:`
`  pass`	

Can you spot the error?

Original code	Human repair

`try:`
`  something()`
`catch AttributeError:`
`  pass`
	`try:`
	`  something()`
	`except AttributeError:`
	`  pass`

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.*

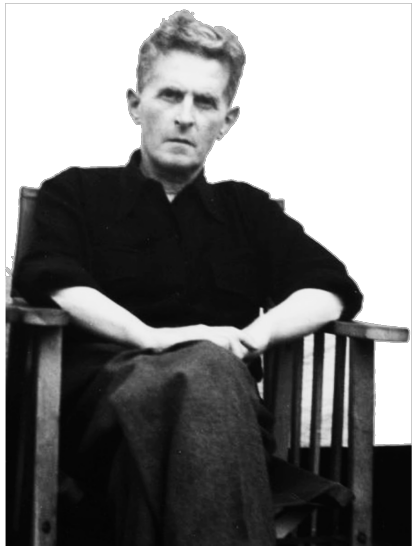
- Often framed as a compromise, *"Let's be pragmatic..."*
- Pragmatism is a principled approach to problem solving.
- Taken seriously, pragmatism is difficult because it requires modeling the needs of multiple stakeholders and balancing competing interests.
- Putting it into practice requires knowing 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 Pierce, 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.



Common sources of syntax errors

- Reading impairments (e.g., dyslexia, dysgraphia)
- Motor impairments (e.g., tremors, Parkinson's)
- Speech impediments (e.g., stuttering, apraxia, Tourette's)
- Visual impairments (e.g., poor eyesight, blindness)
- Language barriers (e.g., foreign and non-native speakers)
- Inexperience (e.g., novice programmers)
- Distraction (e.g., multitasking, fatigue, stress)
- Time pressure (e.g., deadlines, interview coding)
- Inattention (e.g., typographic mistakes, boredom, apathy)
- Lack of feedback (e.g., no syntax highlighting or IDE)

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 $\tilde{\boldsymbol{\sigma}} \in \mathcal{P}\big(\mathcal{L}(\mathcal{G})\big)$.
- 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.

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.

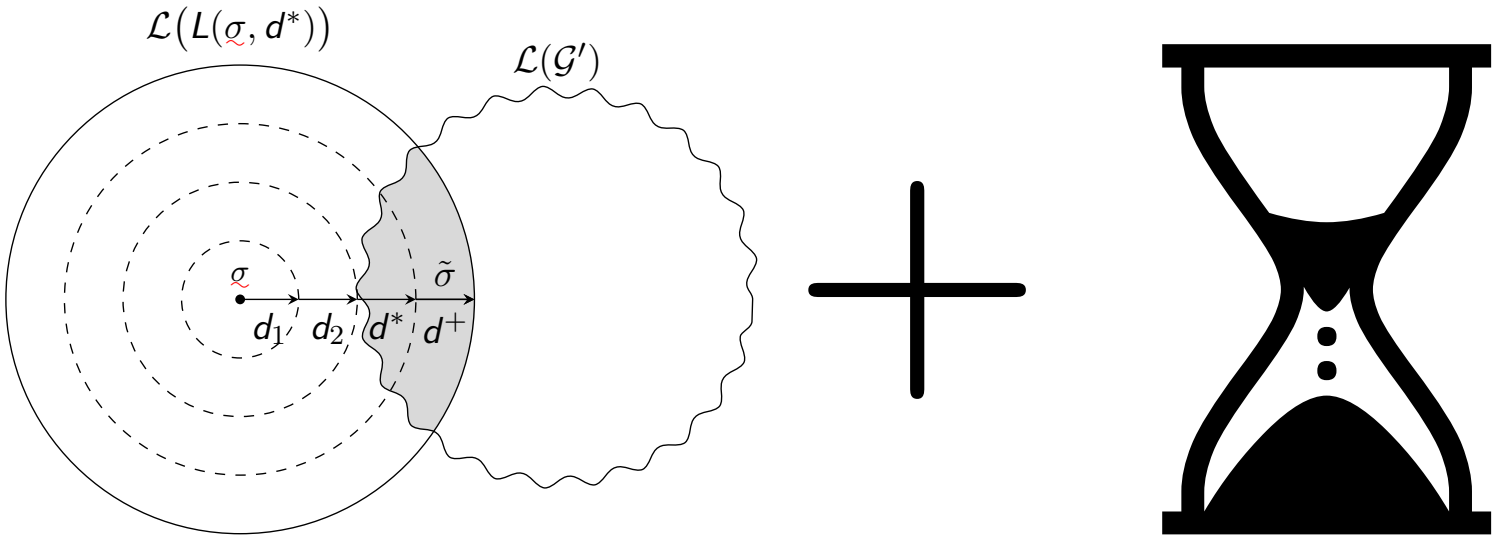
"*Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?*""

Richard Hamming, 1915-1998

Our Contribution

Helps novice programmers fix syntax errors in source code. We do so by solving the **Realtime Bounded Levenshtein Reachability Problem**:

Given a context-free language $\ell : \mathcal{L}(\mathcal{G})$ and an invalid string $\underset{\sim}{\sigma} : \bar{\ell}$, find every syntactically admissible edit $\tilde{\sigma}$ satisfying $\{\tilde{\sigma} \in \ell \mid \Delta(\underset{\sim}{\sigma}, \ell) < r\}$, ranked by a probability metric Δ, under hard realtime constraints.



Natural language: *Rapidly finds syntactically valid edits within a small neighborhood, ranked by tokenwise similarity and statistical likliehood.*

Bounded Levenshtein Reachability

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

Definition (Bounded Levenshtein-CFL reachability)

Given a CFL ℓ and an invalid string $\underset{\sim}{\sigma} : \bar{\ell}$, the BCFLR problem is to find every valid string reachable within d edits of $\underset{\sim}{\sigma}$, i.e., we seek $L(\underset{\sim}{\sigma}, d) \cap \ell$ where $L(\underset{\sim}{\sigma}, d) := \{\sigma \mid \Delta(\underset{\sim}{\sigma}, \sigma) \leq d\}$ and Δ is the Levenshtein metric.

To solve this problem, we will first pose a simpler problem that only considers localized edits, then turn our attention back to BCFLR.

Definition (Porous completion)

Let $\underline{\Sigma} := \Sigma \cup \{\_\}$, 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 $\mathsf{H}(\sigma)$. Given a *porous string*, $\sigma : \underline{\Sigma}^*$ we seek all syntactically admissible inhabitants, i.e., $A(\sigma) := \mathsf{H}(\sigma) \cap \ell$.

Ranked repair under realtime constraints

$A(\sigma)$ is often a very large-cardinality set, so we want a procedure which prioritizes likely repairs first, without exhaustive enumeration. Specifically,

Definition (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,

$$R(\ell_\cap, P_\theta) := \underset{\{\boldsymbol{\sigma} | \boldsymbol{\sigma} \subseteq \ell_\cap, |\boldsymbol{\sigma}| \leq k\}}{\operatorname{argmax}} \sum_{\sigma \in \boldsymbol{\sigma}} \mathrm{P}(\sigma \mid \underset{\sim}{\sigma}, \theta) \tag{1}$$

We want a procedure $\hat{R}$, minimizing $\mathbb{E}_{G,\sigma}\left[D_{\mathrm{KL}}(\hat{R} \parallel R)\right]$ and total latency.

Since R is intractable in general, we want a procedure that approximates it for a representative sampling of natural context-free grammars and strings, i.e., real-world programming languages and source code snippets.

From CFL Reachability to Real World Program Repair

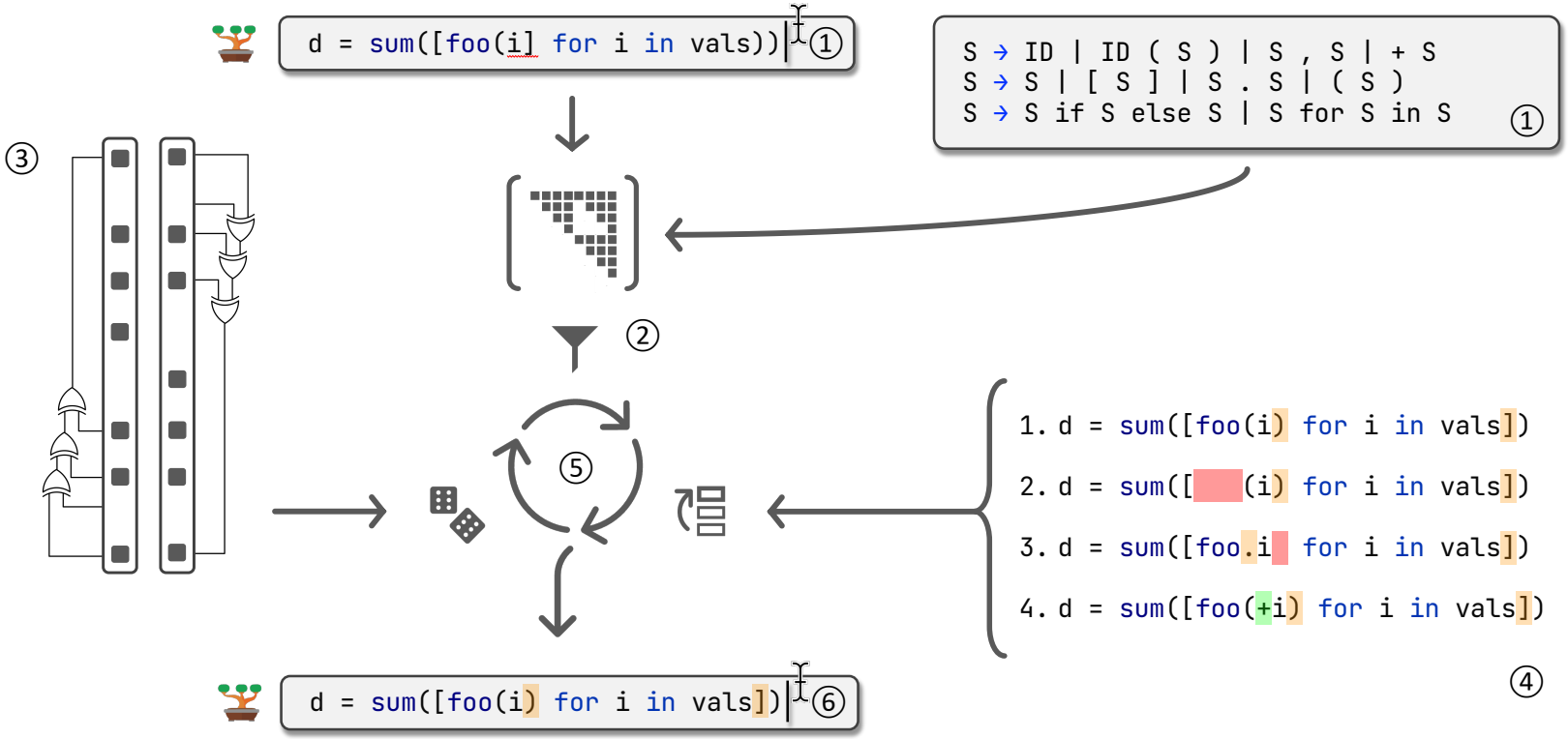
To fix real code, we needed to overcome a variety of interesting challenges:

- **Syntax mismatch**: The syntax of real-world programming languages does not exactly correspond to the theory of formal languages.
- **Source code** $\approx$ **PL**: Most of the time, source code in the wild is incomplete or only loosely approximates a programming language.
- **Responsiveness**: The usefulness of synthetic repairs is inversely proportional to the amount of time required to generate them.
- **Edit generation**: How do we generate edits that are (1) syntactically admissible (2) statistically plausible and (3) semantically meaningful?
- **Evaluation**: Big code and version control is too coarse-grained, contains irrelevant edits, not representative of small errors/fixes.

Design Criteria

- First and foremost, the framework must be **sound** and **complete**. It must (1) accept arbitrary CFGs, and (2) recognize all and only syntactically valid strings within a fixed edit distance of a given string.
- Second, we strongly prefer repairs to be **plausible** and **consistent**, i.e., likely to be expressed in practice and consistent with the author's intent and surrounding context (e.g., stylistically, semantically)
- Third, recommendations must be **efficient** and **responsive**. We must be able to recognize valid strings in subcubic time, and generate repairs in (at worst) polynomial time. These criteria are necessary to provide real-time feedback whilst the user is typing.
- Fourth, the framework must be **robust** and **scalable**. Ceteris paribus, the framework should be robust to multiple errors, handle large grammars and scale linearly with additional processors.

High-level architecture overview



Our framework consists of three components, (2) a solver, (4) ranker and (3) sampler. Given an invalid string and grammar (1), we first compile them into a multilinear system of equations which can be solved directly, yielding a set of repairs that are ranked using a suitable scoring function (4). Optionally, we may introduce stochastic edits to the string using the Levenshtein ball sampler (3) and extract the solutions incrementally (5).

Uniform sampling benchmark on natural syntax errors

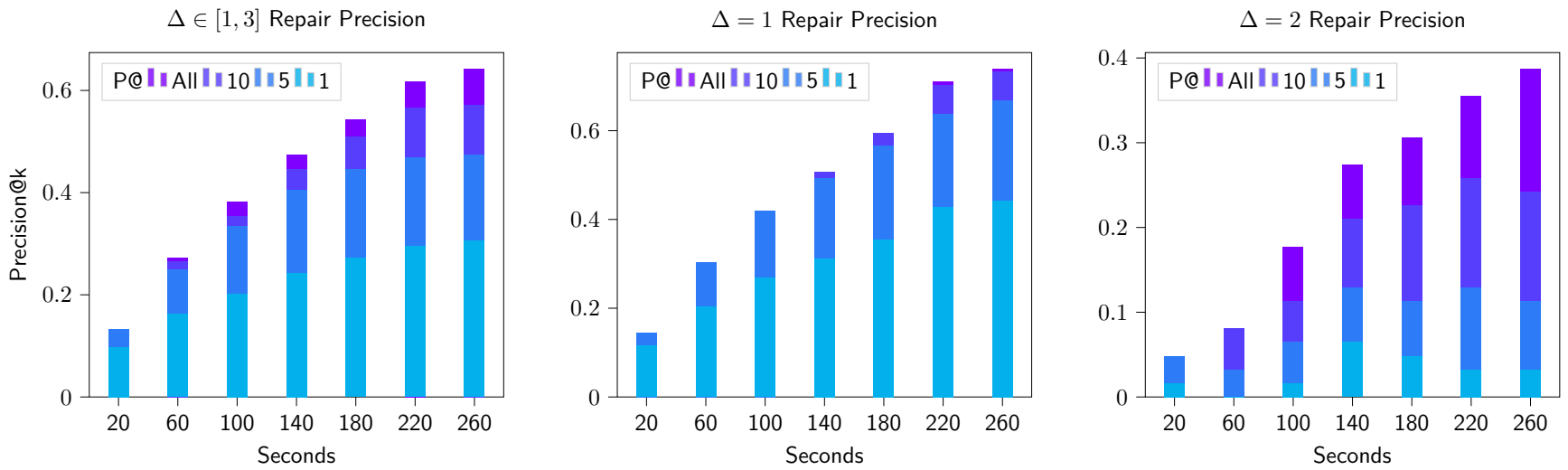


Figure: Repairs discovered before the latency cutoff are reranked based on their tokenwise perplexity and compared for an exact lexical match with the human repair at or below rank k. We note that the uniform sampling procedure is not intended to be used in practice, but provides a baseline for the empirical density of the admissible set, and an upper bound on the latency required to attain a given precision.

Adaptive sampling benchmark on natural syntax errors

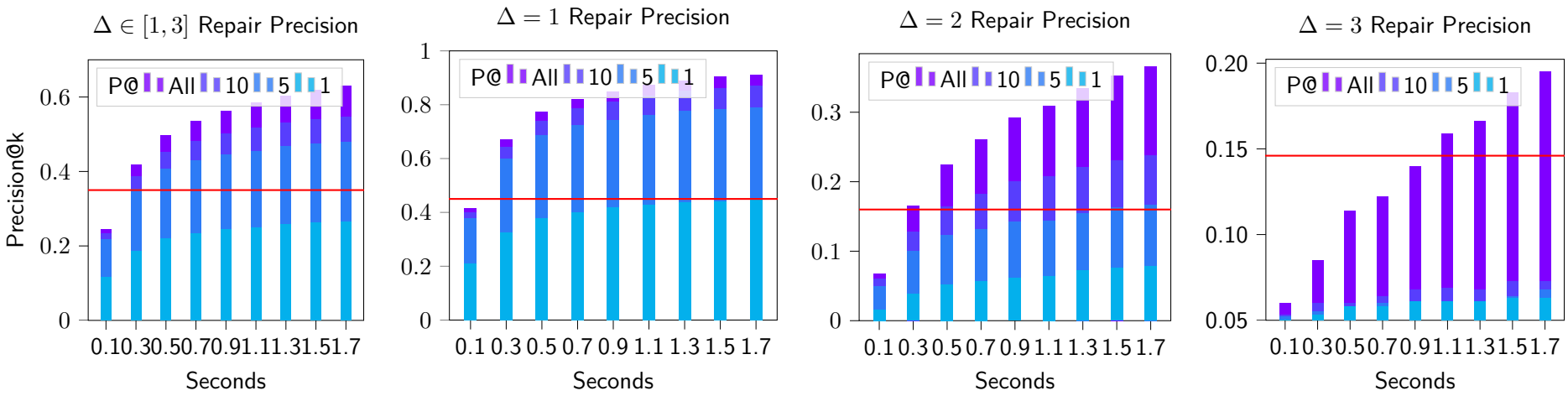


Figure: Adaptive sampling repairs. The red line indicates Seq2Parse precision@1 on the same dataset. Since it only supports generating one repair, we do not report precision@k or the intermediate latency cutoffs.

Precision on Error Correction in Synthetic Language

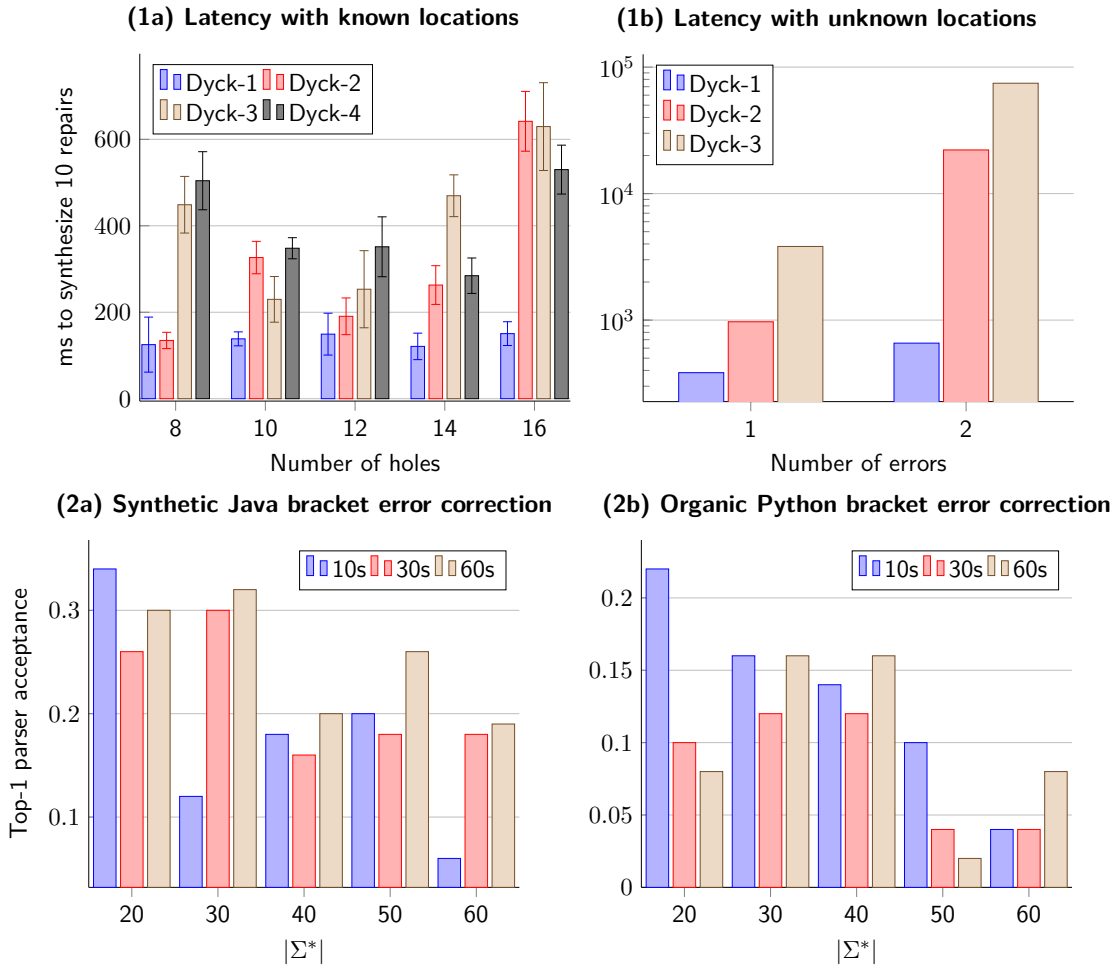


Figure: Benchmarking bracket correction latency and accuracy across two bracketing langauges, one generated from Dyck-n, and the second uses an abstracted source code snippet with imbalanced parentheses.

Uniform sampling benchmark

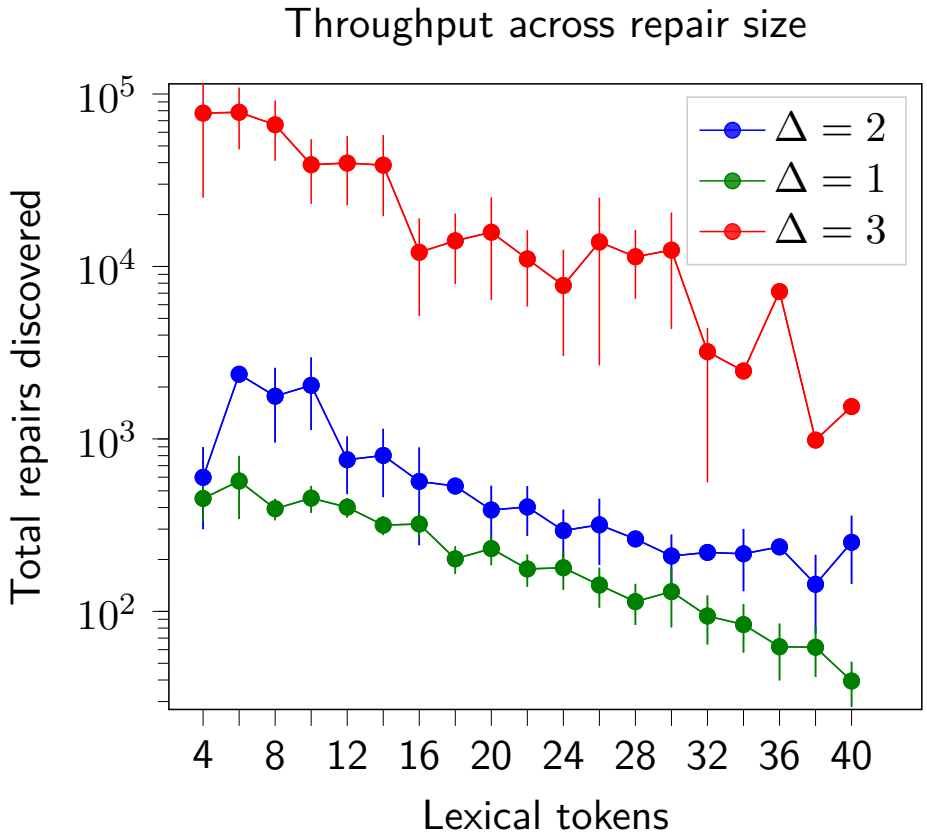
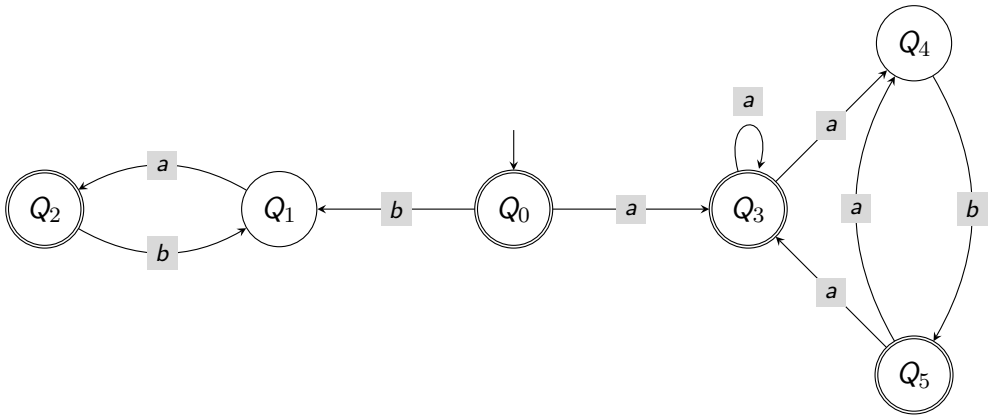


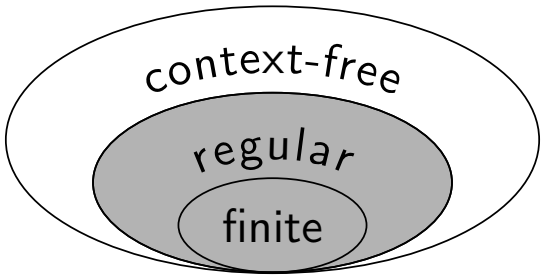
Figure: We evaluate throughput by sampling edits across invalid strings $|\sigma| \leq 40$ from the StackOverflow dataset of varying length, and measure the total number of syntactically valid edits discovered, as a function of string length and language edit distance $\Delta \in [1, 3]$. Each trial is terminated after 10 seconds, and the experiment is repeated across 7.3k total 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)*`:

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



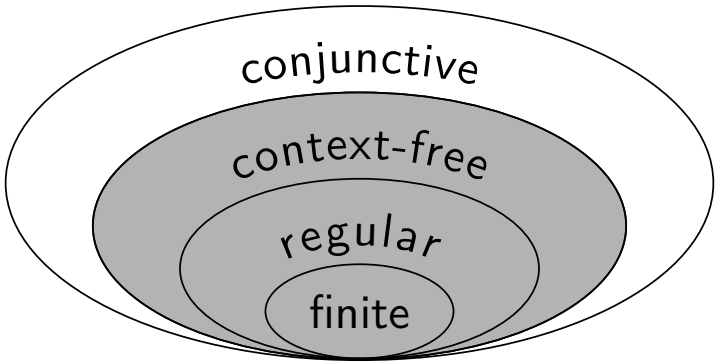


Background: Context-free grammars

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 n^ω, n.b. CFLs are **not** closed under intersection!

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

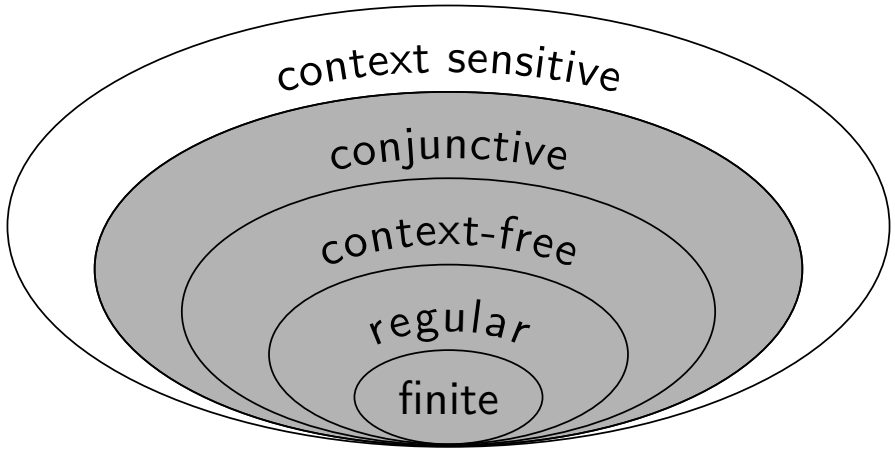
Every CFG has a normal form $P^* : V \times (V^2 \mid \Sigma)$, i.e., every production can be refactored into either $v_0 \to v_1 v_2$ or $v_0 \to \sigma$, where $v_{0\ldots2} : V$ and $\sigma : \Sigma$, e.g., $\{S \to SS \mid (S) \mid ()\} \Leftrightarrow^* \{S \to XR \mid SS \mid LR, L \to (, R \to),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



Background: Conjunctive grammars

Conjunctive grammars naturally extend CFGs with CFL union and intersection, respecting closure under those operations. Equivalent to trellis automata, which are like contractive elementary cellular automata. Language inclusion is decidable in P.

$$\frac{\Gamma \vdash \mathcal{G}_1, \mathcal{G}_2 : \mathtt{CG}}{\Gamma \vdash \exists\, \mathcal{G}_3 : \mathtt{CG}\ .\ \mathcal{L}_{\mathcal{G}_1} \cap \mathcal{L}_{\mathcal{G}_1} \leftrightarrow \mathcal{G}_3}\ \cap$$



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, $\star$ be Kleene star, and $\complement$ be complementation:

	$\cup$	$\cap$	$\cdot$	$\star$	$\complement$
Finite[1]	✓	✓	✓	✓	✓
Regular[1]	✓	✓	✓	✓	✓
Context-free[1]	✓	✗	✓	✓	✗
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.

Context-free parsing, distilled

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

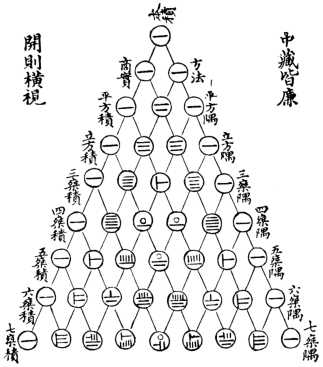
If we define $\sigma_r^{\pitchfork} := \{w \mid (w \to \sigma_r) \in P\}$, then initialize $M^0_{r+1=c}(\mathcal{G}', e) := \sigma_r^{\pitchfork}$ and solve for the fixpoint $M^* = M + M^2$,

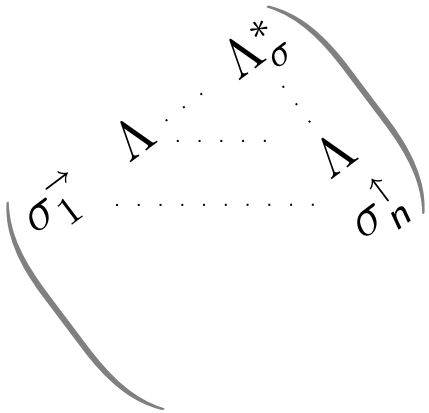
$$M^0 := \begin{pmatrix} \varnothing & \sigma_1^{\rightarrow} & \varnothing & \cdots & \varnothing \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \varnothing \\ \vdots & & & \ddots & \sigma_n^{\uparrow} \\ \varnothing & \cdots & \cdots & \cdots & \varnothing \end{pmatrix}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

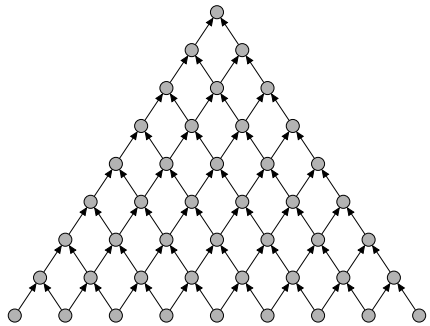
$S \Rightarrow^* \sigma \iff \sigma \in \mathcal{L}(\mathcal{G})$ iff $S \in \Lambda_\sigma^*$, i.e., $\mathbb{1}_{\Lambda_\sigma^*}(S) \iff \mathbb{1}_{\mathcal{L}(\mathcal{G})}(\sigma)$.

Lattices, Matrices and Trellises

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







Jia Xian Triangle
Jia, ∼1030 A.D.

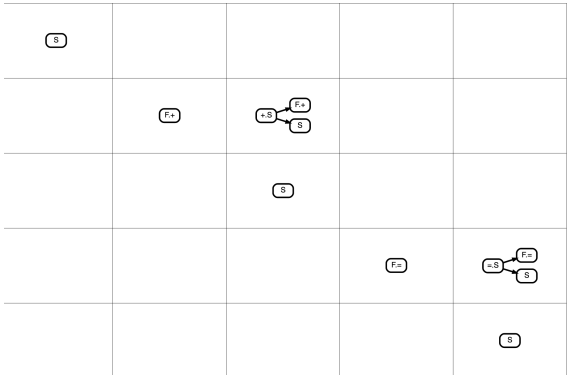
CYK Parsing
Sakai, 1961 A.D.

Trellis Automaton
Dyer, 1980 A.D.

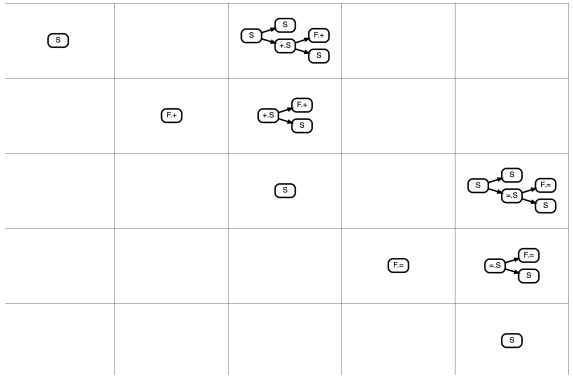
A few observations on algebraic parsing

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

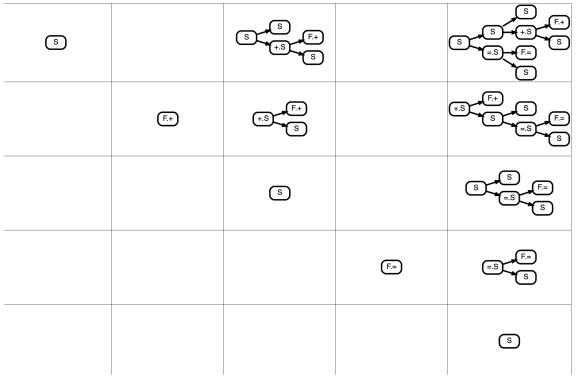
$\mathbf{M}_1 = \mathbf{M}_0 + \mathbf{M}_0^2$



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



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



- 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 (our contribution)

- 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

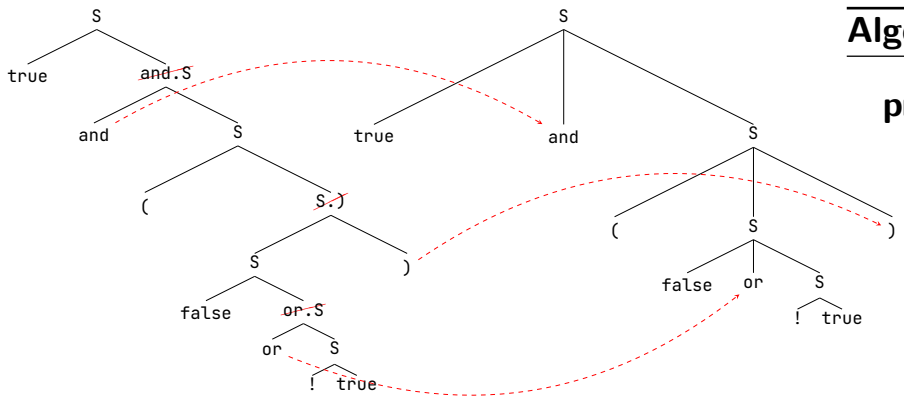
Satisfiability + holes (our contribution)

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

Chomsky Denormalization

Chomksy normalization is needed for matrix-based parsing, however produces lopsided parse trees. We can denormalize them using a simple recusive procedure to restore the natural shape of the original CFG:



Pre-Denormalization Post-Denormalization

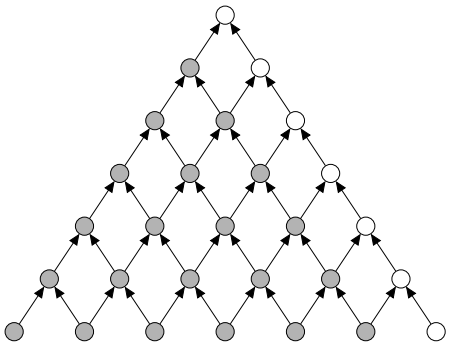
Algorithm Rewrite procedure for tree denormalization

```
procedure DENORMALIZE(t: Tree)
    stems ← { DENORMALIZE(c) | c ∈ t.children }
    if t.root ∈ V_G' \ V_G then
        return stems          ▷ Drop synthetic nonterminals.
    else                      ▷ Graft the denormalized children on root.
        return { Tree(root, stems) }
    end if
end procedure
```

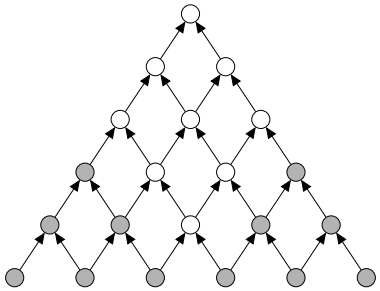
All synthetic nonterminals are excised during Chomsky denormalization. Rewriting improves legibility but does not alter the underlying semantics.

Incremental parsing

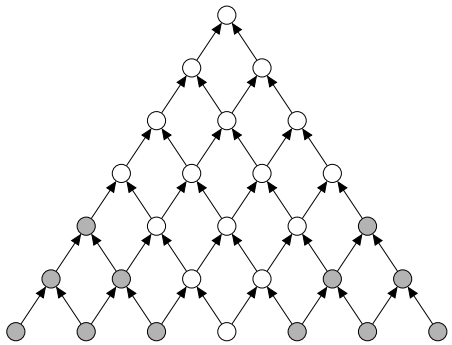
Should only need to recompute submatrices affected by individual edits. In the worst case, each edit requires quadratic complexity in terms of $|\Sigma^*|$, assuming $\mathcal{O}(1)$ cost for each CNF-nonterminal subset join, $\mathbf{V}'_1 \otimes \mathbf{V}'_2$.



Append
$\mathcal{O}(n+1)$



Delete
$\mathcal{O}\left(\frac{1}{4}(n-1)^2\right)$



Insert
$\mathcal{O}\left(\frac{1}{4}(n+1)^2\right)$

Related to **dynamic matrix inverse** and **incremental transitive closure** with vertex updates. With a careful encoding, we can incrementally update SAT constraints as new keystrokes are received to eliminate redundancy.

Conjunctive parsing

It is well-known that the family of CFLs is not closed under intersection. For example, consider $\mathcal{L}_\cap := \mathcal{L}_{\mathcal{G}_1} \cap \mathcal{L}_{\mathcal{G}_2}$:

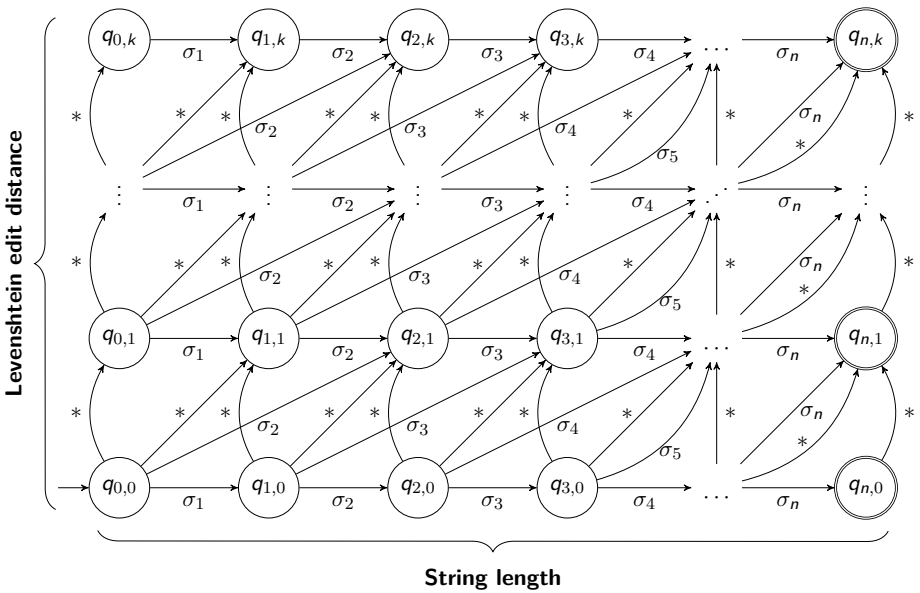
$$P_1 := \left\{ S \to LR, \quad L \to ab \mid aLb, \quad R \to c \mid cR \right\}$$
$$P_2 := \left\{ S \to LR, \quad R \to bc \mid bRc, \quad L \to a \mid aL \right\}$$

Note that $\mathcal{L}_\cap$ generates the language $\left\{ a^d b^d c^d \mid d > 0 \right\}$, which according to the pumping lemma is not context-free. To encode $\mathcal{L}_\cap$, we intersect all terminals $\Sigma_\cap := \bigcap_{i=1}^{c} \Sigma_i$, then for each $t_\cap \in \Sigma_\cap$ and CFG, construct an equivalence class $E(t_\cap, \mathcal{G}_i) = \{w_i \mid (w_i \to t_\cap) \in P_i\}$ as follows:

$$\bigwedge_{t \in \Sigma_\cap} \bigwedge_{j=1}^{c-1} \bigwedge_{i=1}^{|\sigma|} E(t_\cap, \mathcal{G}_j) \equiv_{\sigma_i} E(t_\cap, \mathcal{G}_{j+1}) \tag{2}$$

Levenshtein reachability and monotone infinite automata

MAIA



CFG

$$S \Rightarrow \{\cdot \in Q \mid \delta(\cdot, q_{n,0}) \leq k\}$$

$$* \Rightarrow \{\cdot \in \Sigma\}$$

$$\{q_{i,j} \Rightarrow \{q_{i,j-1}*\} \mid i,j \in [1,n] \times [1,k]\}$$

$$\{q_{i,j} \Rightarrow \{q_{i-1,j-1}*\} \mid i,j \in [1,n] \times [1,k]\}$$

$$\{q_{i,j} \Rightarrow \{q_{i-1,j}\sigma_i\} \mid i,j \in [1,n] \times [0,k]\}$$

$$\{q_{i,j} \Rightarrow \{q_{i-2,j-1}\sigma_i\} \mid i,j \in [2,n] \times [1,k]\}$$

Figure: Bounded Levenshtein reachability from $\sigma : \Sigma^n$ is expressible as either a monotone acyclic infinite automata (MAIA) populated by accept states within radius k of $S = q_{n,0}$ (left), or equivalently, a left-linear CFG whose productions bisimulate the transition dynamics up to a fixed horizon (right), accepting only strings within Levenshtein radius k of σ.

The Chomsky-Levenshtein-Bar-Hillel Construction

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

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

The Levenshtein automata is another kind of lattice, not in the order-theoretic sense, but in the automata-theoretic sense.

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

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

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

Semiring algebras: Part I

There are a number of alternate semirings which can be used 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 II

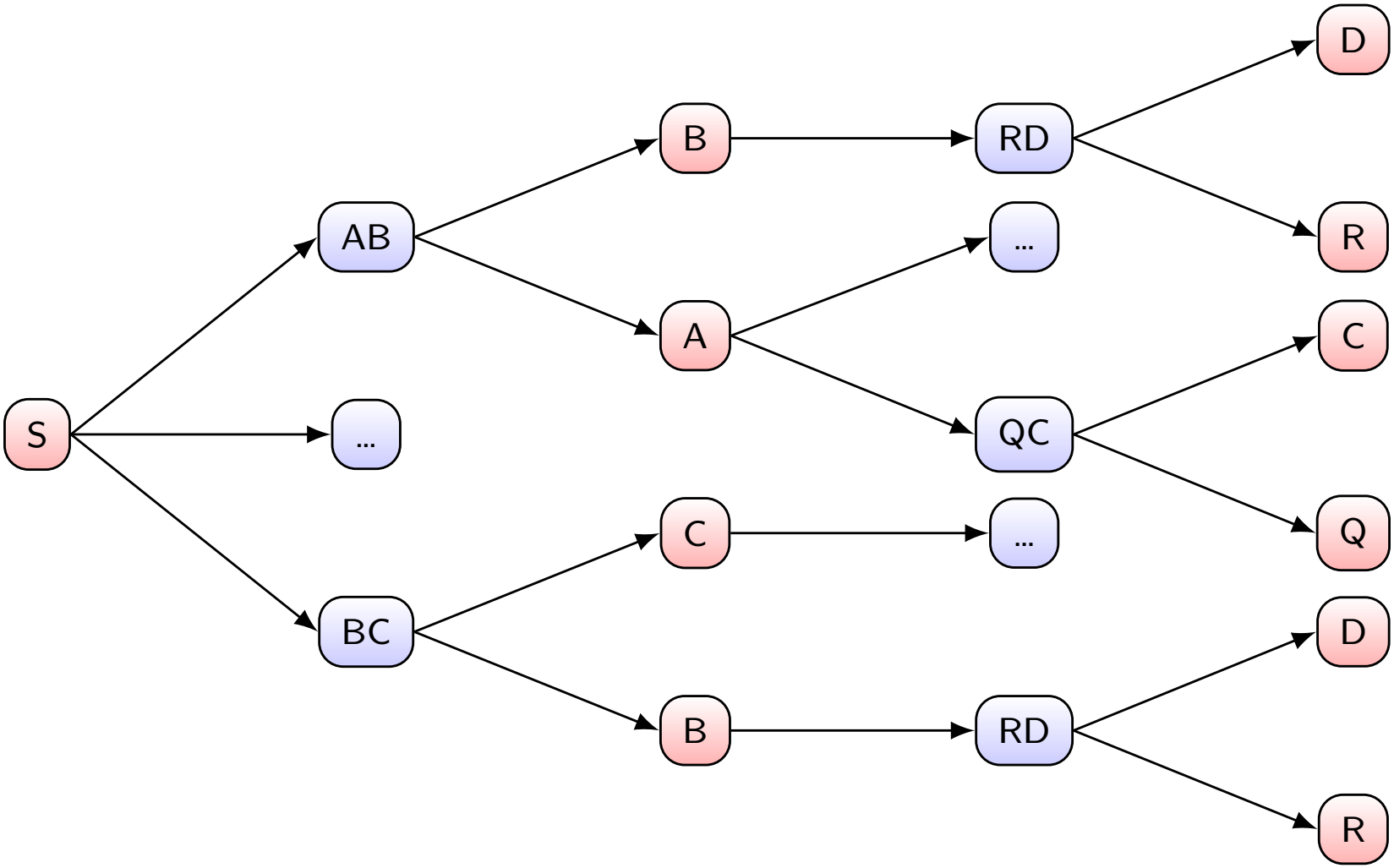


Figure: A partial $\mathbb{T}_2$ for the grammar with productions $P = \{S \to BC \mid \ldots \mid AB, B \to RD \mid \ldots, A \to QC \mid \ldots\}$.

A pairing function for sampling bounded binary trees

The type $\mathbb{T}_2$ of all trees that can be generated by a CNF CFG is a nested datatype: $P(a) = 1 + aL(P(a)^2), L(p) = 1 + pL(p)$. We can count and sample the total number of trees induced by a given sketch template as:

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

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

Error Correction: Levenshtein q-Balls

Now that we have a reliable method to fix *localized* errors, $S : \mathcal{G} \times (\Sigma \cup \{\varepsilon, \_\})^n \to \{\Sigma^n\} \subseteq \mathcal{L}_\mathcal{G}$, given some unparseable string, i.e., $\sigma_1 \dots \sigma_n : \Sigma^n \cap \mathcal{L}_\mathcal{G}^\complement$, where should we put holes to obtain a parseable $\sigma' \in \mathcal{L}_\mathcal{G}$? One way to do so is by sampling repairs, $\boldsymbol{\sigma} \sim \Sigma^{n \pm q} \cap \Delta_q(\sigma)$ from the Levenshtein q-ball centered on σ, i.e., the space of all admissible edits with Levenshtein distance $\leq q$ (this is loosely analogous to a finite difference approximation). To admit variable-length edits, we first add an ε^+-production to each unit production:

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

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

Error Correction: d-Subset Sampling

Next, suppose $U : \mathbb{Z}_2^{m \times m}$ is a matrix whose structure is shown in Eq. 3, wherein C is a primitive polynomial over $\mathbb{Z}_2^m$ with coefficients $C_{1 \ldots m}$ and semiring operators $\oplus := \veebar, \otimes := \wedge$:

$$U^t V = \begin{pmatrix} C_1 & \cdots & C_m \\ \top & \circ & \circ \\ \circ & \ddots & \vdots \\ \vdots & \ddots & \vdots \\ \circ & \circ \; \top & \circ \end{pmatrix}^t \begin{pmatrix} V_1 \\ \vdots \\ \vdots \\ V_m \end{pmatrix} \tag{3}$$

Since C is primitive, the sequence $\mathbf{S} = (U^{0 \ldots 2^m - 1} V)$ must have *full periodicity*, i.e., for all $i, j \in [0, 2^m)$, $\mathbf{S}_i = \mathbf{S}_j \Rightarrow i = j$. To uniformly sample $\boldsymbol{\sigma}$ without replacement, we first form an injection $\mathbb{Z}_2^m \rightharpoonup \left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^{2d}$ using a combinatorial number system, cycle over $\mathbf{S}$, then discard samples which have no witness in $\left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^{2d}$. This method requires $\widetilde{\mathcal{O}}(1)$ per sample and $\widetilde{\mathcal{O}}\left(\binom{n}{d} |\Sigma + 1|^{2d} \right)$ to exhaustively search $\left\{ {n \atop d} \right\} \times \Sigma_\varepsilon^{2d}$.

Error Correction: Sketch Templates

Finally, to sample $\boldsymbol{\sigma} \sim \Delta_q(\underset{\sim}{\sigma})$, we enumerate a series of sketch templates $H(\sigma, i) = \sigma_{1\ldots i-1}\ \_\ \_\ \sigma_{i+1\ldots n}$ for each $i \in \cdot \in \binom{n}{d}$ and $d \in 1 \ldots q$, then solve for $\mathcal{M}^*_{\boldsymbol{\sigma}}$. If $S \in \Lambda^*_{\boldsymbol{\sigma}}?$ has a solution, each edit in each $\sigma' \in \boldsymbol{\sigma}$ will match exactly one of the following seven edit patterns:

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

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

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

An Simple Reachability Proof

Lemma

For any nonempty language $\ell : \mathcal{L}(\mathcal{G})$ and invalid string $\underset{\sim}{\sigma} : \Sigma^n$, there exists an $(\tilde{\sigma}, m)$ such that $\tilde{\sigma} \in \ell \cap \Sigma^m$ and $0 < \Delta(\underset{\sim}{\sigma}, \ell) \leq \max(m, n) < \infty$, where Δ denotes the Levenshtein edit distance.

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 $m := |\tilde{\sigma}|$. 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., $0 < \Delta(\underset{\sim}{\sigma}, \ell) < \infty$. □

Probabilistic repair generation

Algorithm Probabilistic reachability

Require: $\mathcal{G}$ grammar, $\underset{\sim}{\sigma}$ broken string, p process ID, c total CPU cores, t_{total} timeout.

$\mathcal{Q} \leftarrow \varnothing, \mathcal{R} \leftarrow \varnothing, \epsilon \leftarrow 1, i \leftarrow 0, Y \sim \mathbb{Z}_2^m, t_0 \leftarrow t_{\text{now}}$ ▷ Initialize replay buffer $\mathcal{Q}$ and reservoir $\mathcal{R}$.
repeat
  if $\mathcal{Q} = \varnothing$ or **Rand**(0, 1) $< \epsilon$ **then**
    $\hat{\sigma} \leftarrow \varphi^{-1}\left(\langle \kappa, \rho \rangle^{-1}(U^{ci+p}Y), \underset{\sim}{\sigma}\right), i \leftarrow i + 1$ ▷ Sample WoR using the leapfrog method.
  else
    $\hat{\sigma} \sim \mathcal{Q} +$ **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**
    $\hat{\sigma} \leftarrow \operatorname{argmin}_{\hat{\sigma} \in \mathcal{R}} PP(\hat{\sigma})$ ▷ 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
  $\epsilon \leftarrow$ **Schedule**$\left((t_{\text{now}} - t_0)/t_{\text{total}}\right)$ ▷ Update exploration/exploitation rate.
until t_{total} elapses.
return $\tilde{\sigma} \in \mathcal{Q}$ ranked by $PP(\tilde{\sigma})$.

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
- Earley (1968) - top-down dynamic programming (no CNF needed)
- 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**

Special thanks

Jin Guo, Xujie Si
Brigitte Pientka, David Yu-Tung Hui,
Ori Roth, Younesse Kaddar, Michael Schröder
Torsten Scholak, Matthew Sotoudeh, Paul Zhu

McGill UNIVERSITY

Mila

Learn more at:

https://tidyparse.github.io