All You Need is Generalized GLL

Parsing algorithms are not only about string processing

Semyon Grigorev

December 29, 2022

Generalized Parsing

- Handles arbitrary context-free grammars
  - Ambiguous
  - (Hidden) Left-recursive (for LL)
- Linear for unambiguous grammars
- Cubic in worst case

Generalized Parsing

- Handles arbitrary context-free grammars
  - Ambiguous
  - (Hidden) Left-recursive (for LL)
- Linear for unambiguous grammars
- Cubic in worst case

- ANTLR: ALL(*)
- Tree-Sitter: GLR
- SDF3 (Spoofax): GLR
- ...

Generalized Parsing

- Handles arbitrary context-free grammars
  - Ambiguous
  - (Hidden) Left-recursive (for LL)
- Linear for unambiguous grammars
- Cubic in worst case

- ANTLR: ALL(*)
- Tree-Sitter: GLR
- SDF3 (Spoofax): GLR
- ...

- GSS — Graph Structured Stack, compact representation of stack
- SPPF — Shared Packed Parse Forest, compact representation of parse forest

Generalized LL (GLL)[1]

- Descriptor
  - Grammar slot: $S \rightarrow a \cdot Sb$
  - Input position
  - Top element of stack
  - Root of tree

[1] A. Afroozeh, A. Izmaylova. "Practical General Top-Down Parsers". 2019.

Generalized LL (GLL)[1]

- Descriptor
  - Grammar slot: $S \to a \cdot Sb$
  - Input position
  - Top element of stack
  - Root of tree

- Step — handle one descriptor
- Step produces a **set** of new descriptors
- Each descriptor handle once
  - Set of handled descriptors
  - Order does not matter

[1] A. Afroozeh, A. Izmaylova. "Practical General Top-Down Parsers". 2019.

Generalized LL (GLL)[1]

- Descriptor
  - Grammar slot: $S \to a \cdot Sb$
  - Input position
  - Top element of stack
  - Root of tree
- Step — handle one descriptor
- Step produces a **set** of new descriptors
- Each descriptor handle once
  - Set of handled descriptors
  - Order does not matter
- Iguana — GLL-based parser generator

[1] A. Afroozeh, A. Izmaylova. "Practical General Top-Down Parsers". 2019.

EBNF Without Transformation

$S \to \varepsilon \mid Sa$ | $S \to a^*$

EBNF Without Transformation

$S \to \varepsilon \mid Sa$

$S \to \varepsilon \mid aSbS \mid cSdS$

$S \to a^*$

$S \to (aSb \mid cSd)^*$

EBNF Without Transformation

$$S \to \varepsilon \mid Sa \qquad\qquad S \to a^*$$

$$S \to \varepsilon \mid aSbS \mid cSdS \qquad\qquad S \to (aSb \mid cSd)^*$$

- Slot — state of DFA (right part of rule)

EBNF Without Transformation

$$S \to \varepsilon \mid Sa \qquad\qquad S \to a^*$$

$$S \to \varepsilon \mid aSbS \mid cSdS \qquad\qquad S \to (aSb \mid cSd)^*$$

- Slot — state of DFA (right part of rule)
- Careful handling of descriptor
  - Both terminal and nonterminal can be the next symbols for the current state at the same time
  - Multiple different nonterminals can be the next symbols for the current state
  - State can be final and contains outgoing edges

EBNF Without Transformation

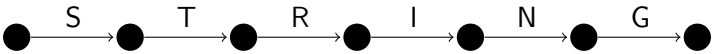
$$S \to \varepsilon \mid Sa \qquad\qquad S \to a^*$$

$$S \to \varepsilon \mid aSbS \mid cSdS \qquad\qquad S \to (aSb \mid cSd)^*$$

- Slot — state of DFA (right part of rule)
- Careful handling of descriptor
  - Both terminal and nonterminal can be the next symbols for the current state at the same time
  - Multiple different nonterminals can be the next symbols for the current state
  - State can be final and contains outgoing edges
- SPPF becomes more complex

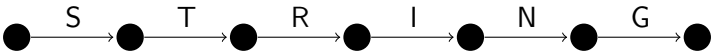
Nonlinear Input Handling

String = linear graph



Nonlinear Input Handling

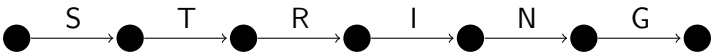
String = linear graph



What makes a graph **not** a string?

Nonlinear Input Handling

String = linear graph

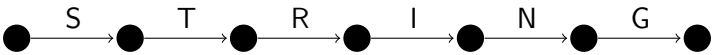


What makes a graph **not** a string?

Branches

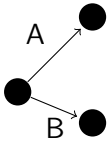
Nonlinear Input Handling

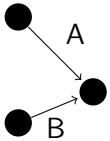
String = linear graph



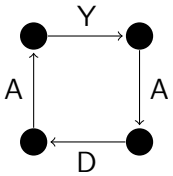
What makes a graph **not** a string?

Branches



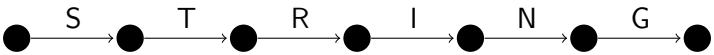


Cycles



Nonlinear Input Handling

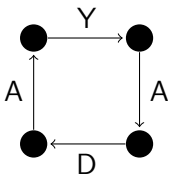
String = linear graph



What makes a graph **not** a string?

Branches

Cycles



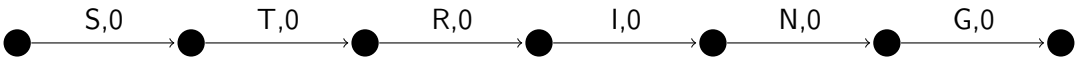
- Multiple different terminals can be the next symbols for the current state
- Each descriptor handle once (cycles and merges not a problem)

Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal

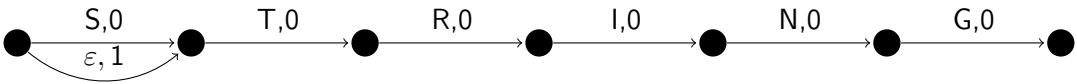
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



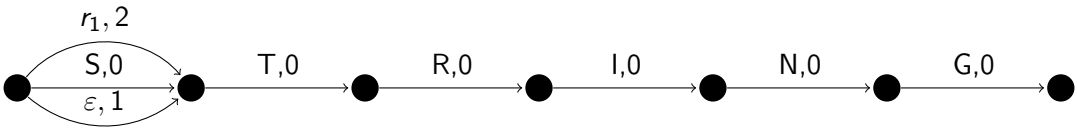
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



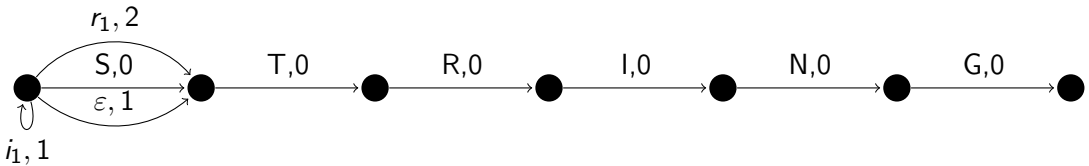
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



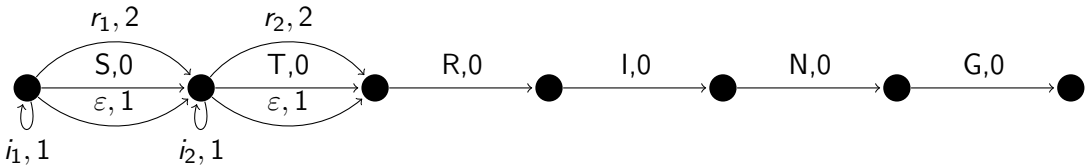
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



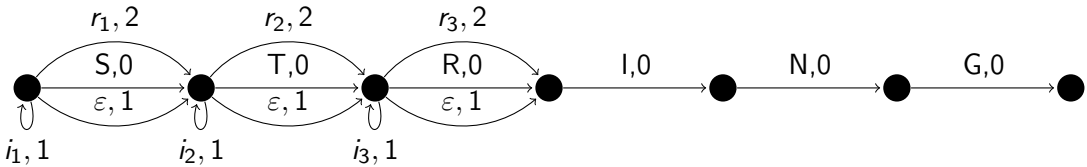
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



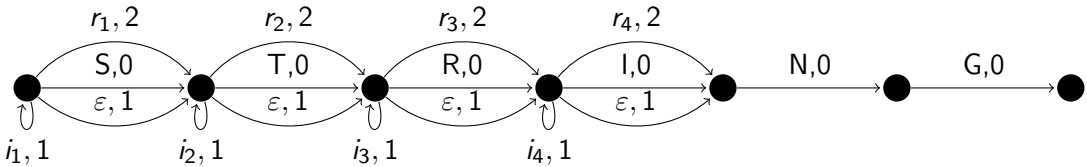
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



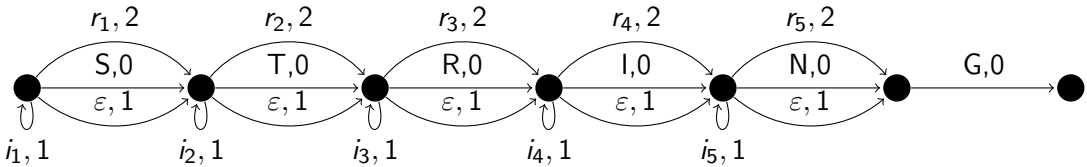
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



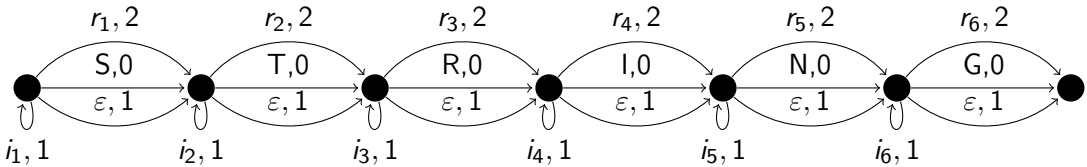
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



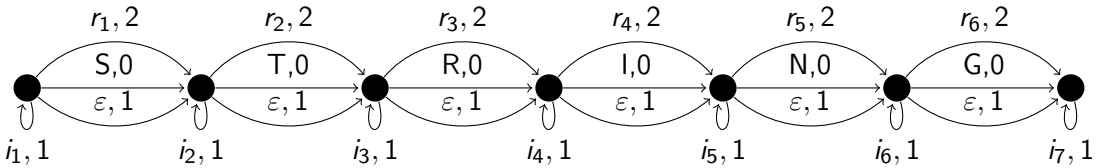
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



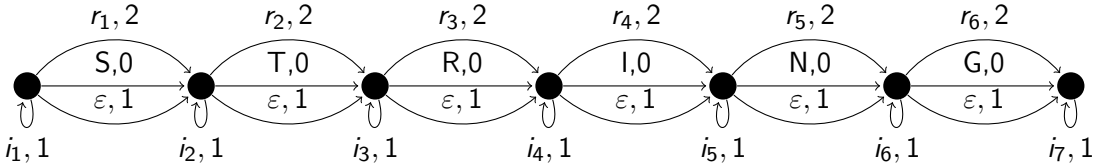
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



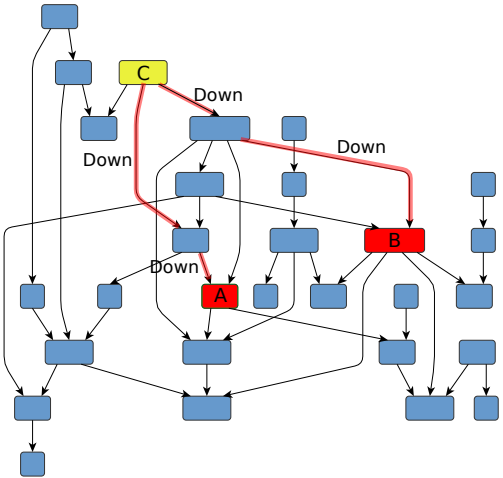
Error Recovery

- Language Editing Distance (LED): for the given language $\mathcal{L}$ and the given string w to find string w_1 such that $w_1 \in \mathcal{L}$, $\texttt{dist}(w, w_1)$ is minimal



- Error recovery $\rightarrow$ find a minimal weight path from start to final in the given graph which forms a word in the given language
- Set of descriptors to handle $\rightarrow$ priority queue

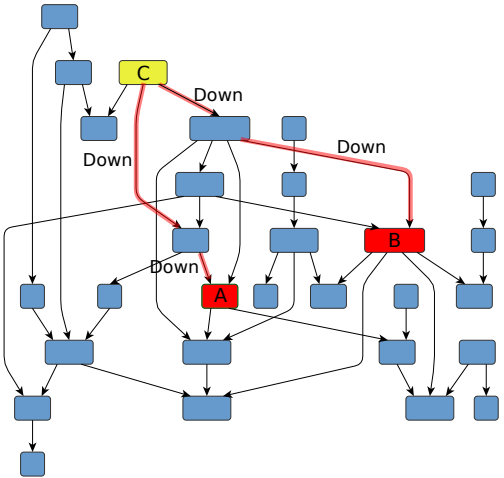
Path Querying



Context-free languages as constraints (CFPQ, Context-Free Path Querying)

- Are nodes A and B on the same level of hierarchy?
- Is there a path of form $\overline{\textbf{Down}}^n\,\textbf{Down}^n$ between A and B?
- Context-free grammar:
  $SameLvl \rightarrow \overline{Down}\ SameLvl\ Down \mid \varepsilon$

Path Querying



Context-free languages as constraints (CFPQ, Context-Free Path Querying)

- Are nodes A and B on the same level of hierarchy?
- Is there a path of form $\overline{\textbf{Down}}^n\,\textbf{Down}^n$ between A and B?
- Context-free grammar:
  $SameLvl \rightarrow \overline{Down}\ SameLvl\ Down \mid \varepsilon$

Applications

- Static code analysis [T. Reps, et al, 1995]
- Graph segmentation [H. Miao, et al, 2019]
- Biological data analysis [P. Sevon, et al, 2008] ...

Conclusion

- Generalized parsing opens the door to
  - Arbitrary context-free grammars handling
  - Natural error recovery
  - Graph navigation
  - ...