Emacs Tree-sitter

Tuấn-Anh Nguyễn

November 25, 2020

The Problem

Like other editors, Emacs relies on regular expressions for many programming functionalities. They are:

- Slow and inaccurate
- Hard to write and read
- Not able to deal with complex syntaxes

If only Emacs had structural understanding of source code, like other IDEs...

Existing Solutions

Language-specific parsers in ELisp

- Not performant
- Hard to maintain
- No generic APIs

Language Server Protocol

- High latency
- Resource-intensive
- Additional dependencies for each language

Tree-sitter

- Parser generator and incremental parsing library
- Originated from Atom, also being integrated into NeoVim
- Used by GitHub for source code analysis and navigation features
- Written in C, targetting all major platforms (and WASM)

Tree-sitter Features

- Fast: incremental parsing, structural sharing
- Uniform: same programming interface across languages
- No dependencies: self-contained, embeddable C code
- Robust error recovery

emacs-tree-sitter

- Tree-sitter bindings for Emacs
- Provides compiled binaries for 3 major platforms (x86_64): macOS, Linux, Windows
- Comprises of 3 packages:
  - `tree-sitter`: base "framework"
  - `tree-sitter-langs`: language bundle
  - `tsc`: core APIs (implicit dependency)

The Foundation: `tree-sitter-mode`

- The base minor mode for other major/minor modes to build on
- Uses Emacs's change tracking hooks for incremental parsing
- Provides an always-up-to-date syntax tree

Syntax Highlighting: `tree-sitter-hl-mode`

- Built in top of `tree-sitter-mode`
- Overrides `font-lock-mode`
- Query-driven

`font-lock-mode`	`tree-sitter-hl-mode`
`font-lock-defaults` `font-lock-add-keywords`	`tree-sitter-hl-default-patterns` `tree-sitter-hl-add-patterns`
Regular expressions Face prefix: `font-lock-`	Lisp-like query patterns Face prefix: `tree-sitter-hl:`

Core APIs: `tsc`

- Parsing: generic parser object, specific language objects
- Inspecting: node type, location, errors, related nodes
- Walking: efficient tree traversal through a cursor object
- Querying: searching for structural patterns with a sexp query

Pattern Matching with Tree Queries

- Structural patterns written in a Lisp-like syntax
- Matching by node types, field names, predicates
- Capturing nodes for further processing
- Alternations, repetitions, wildcards

Language Bundle: `tree-sitter-langs`

- A package that provides compiled grammar binaries and highlighting queries for several languages. It currently bundles C, C++, CSS, Go, HTML, Java, JavaScript, PHP, Python, Ruby, Rust, TypeScript.
- This should be treated as an interim distribution mechanism that helps bootstrapping `tree-sitter`'s adoption. Eventually, these files should be provided by language major modes themselves.

Areas for Contribution

- Maintaining syntax highlighting queries
- Integrating `tree-sitter` into an existing major mode, or writing a new one
- Writing minor modes and integration packages: `imenu`, `evil`, `xref`, `hideshow`, `polymode`, ...
- Improving language grammars: `https://github.com/tree-sitter/`

Writing Language Grammars

- GLR with an DSL embedded in JavaScript
- Generation-time conflicts are resolved by precedences
- Parse-time conflicts are resolved by dynamic precedences

Links

- Source code:
  https://github.com/ubolonton/emacs-tree-sitter/
- Documentation:
  https://ubolonton.github.io/emacs-tree-sitter/
- Tree-sitter's documentation:
  https://tree-sitter.github.io/tree-sitter/
- Tree-sitter's StrangeLoop talk:
  https://youtu.be/Jes3bD6P0To

Addendum: Dynamic Modules

The dynamic module that powers `tree-sitter` is written in Rust. Overall it's a much nicer experience than doing the same in C. There are various areas where Emacs's dynamic module support can be improved:

- Direct read-only access to strings and buffer texts
- Better printed representation for `user-ptr` objects
- Module-defined `equal` for `user-ptr` (or sensible defaults)
- Direct calling of core C functions to avoid `funcall` overhead
- Releasing/acquiring the GIL