Semantic Highlighting in TLA+

How to create modern user-assistive language tooling using tree-sitter

Web demo: https://tlaplus-community.github.io/tree-sitter-tlaplus/
Repo: https://github.com/tlaplus-community/tree-sitter-tlaplus

Andrew Helwer
https://ahelwer.ca
consulting@disjunctive.llc

What can you do with real-time access to the TLA+ parse tree?

```
---- MODULE DistributedKVS ----
CONSTANTS Key, Value, Node
VARIABLE state

TypeOK ≜
  state ∈ [Node → [Key → Value]]

====
```

```
(source_file (module (header_line) (identifier) (header_line)
  (constant_declaration (identifier) (identifier) (identifier))
  (variable_declaration (identifier) (identifier))
  (operator_definition (identifier) (def_eq)
    (bound_infix_op (identifier_ref) (in)
      (set_of_functions
        (identifier_ref) (maps_to)
        (set_of_functions
          (identifier_ref) (maps_to) (identifier_ref)
        )
      )
    )
  )
(double_line)))
```

Tree-Sitter Grammar Capabilities

- Error recovery
- Fast incremental parsing
- Tree queries
- Standalone library
- Bindings for many languages
- Standardized API – and there are tree-sitter grammars for many languages!

Tree-Sitter Grammars vs. Language Servers

Language servers:

- Standardized LSP capability-based API (get reference, go to definition, etc.)
- Written in any language, framework, runtime, etc.
- LSP clients built into many platforms (VS Code, Neovim, Eclipse, Emacs)

Tree-Sitter grammars:

- Standardized API for parse/edit/update, tree query, and tree traversal
- Generated C code from DSL, plus C/C++ external scanner
- Supported by fewer platforms so far (Neovim, GitHub)
- People have built very interesting apps that consume tree-sitter grammars!

Some Ideas

- **Semantic highlighting**

  "Syntax highlighting is a waste of an information channel" – Hillel Wayne

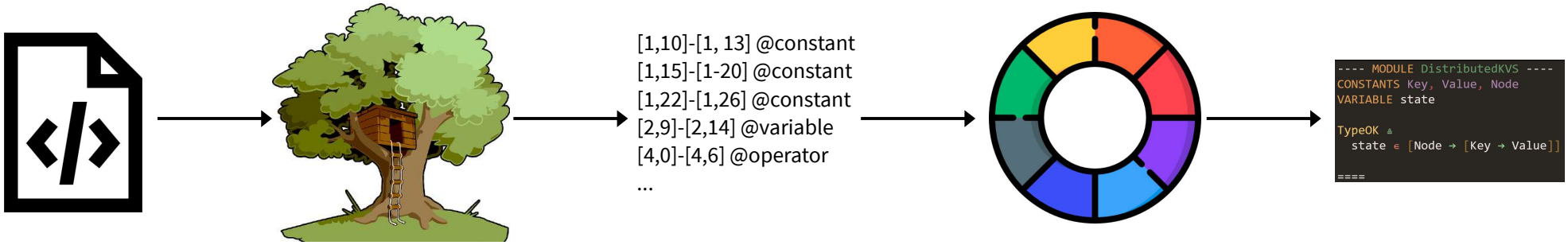
- **Code folding**
- **Symbol/reference finding**
- **Code analysis/linting/transformation tools**
- **Accessibility: coding via alternative input methods**

  https://github.com/pokey/cursorless-vscode

Highlighting

- Conventional syntax highlighting uses regular expressions
- Tree-sitter enables exposing semantic information via highlighting
  - the ∈ in **x ∈ Nat** can be highlighted differently from the one in ∀ **x ∈ Nat : …**
- Identifiers can be highlighted based on what sort of thing they refer to
- Conjunction/disjunction lists can be given “rainbow” highlighting
- Can highlight expressions of the form `other ≜ CHOOSE v : v ∉ S`
- All powered by tree queries!

How Highlighting Works



Tree Queries

```
(source_file (module (header_line) (identifier) (header_line)
  (constant_declaration (identifier) (identifier) (identifier))
  (variable_declaration (identifier) (identifier))
  (operator_definition name: (identifier) (def_eq)
    (bound_infix_op (identifier_ref) (in)
      (set_of_functions
        (identifier_ref) (maps_to)
        (set_of_functions
          (identifier_ref) (maps_to) (identifier_ref)
        )
      )
    )
  )
(double_line)))
```

```
(operator_definition name: (identifier) @operator)
```

Tree Queries

Basic node match: (node_name) @capture_name

Match node with child: (operator_definition definition: (conj_list)) @capture

Negation, quantification, wildcards, predicates, etc.

https://tree-sitter.github.io/tree-sitter/using-parsers#pattern-matching-with-queries

It’s really easy to define highlighting based on captures!

Consuming the Grammar

Official bindings available for Python, TypeScript/JavaScript, Rust, and C/C++

Many other unofficial bindings to be found

Grammar available as NPM module, Rust crate, and as download from GitHub

https://github.com/tlaplus-community/tlaplus-tool-dev-examples

How can I actually use the grammar today?

Install Neovim

Install/enable nvim-treesitter plugin

:TSInstall tlaplus

Open any .tla file in nvim

Unicode TLA+

Benefits:

- Get to see all the beautiful math symbols as you're writing the spec!

Drawbacks:

- Requires care in choice of font
- SANY/TLC don't support it (Ron Pressler has worked on this)
- Tricky to convert between ASCII & Unicode TLA+ specs

https://github.com/tlaplus-community/tlaplus-standard/tree/main/unicode

https://github.com/tlaplus-community/tlaplus-nvim-plugin

Fun nooks & crannies to the TLA+ language

- The block comment start token (* is a valid character sequence in the language
  - Passing multiplication operator as a parameter to another operator, as in `f(*)`
- You can define values in binary, octal, or hex with `\b0101`, `\o5678`, `\hF85A`
- You can define RECURSIVE operators inside LET/IN constructs
- You can use LET/IN constructs anywhere you can have an expression
- You can "destructure" tuples, as in `f[⟨a,b,c⟩ ∈ Nat×Nat×Nat] ≜ a`
- Complicated ambiguity with the `(+)` infix operator; what does `f (+) g` mean?
- Ability to refer to sub-expressions of larger expressions
- You can write <= as =<

The experience of writing a TLA+ parser

Conjunction/disjunction lists are difficult to parse!

Proofs are also difficult to parse!

CASE constructs are difficult to parse!

I started out thinking TLA+ was a small language, but actually it is complicated!

How can you contribute?

Add your specs to https://github.com/tlaplus/examples

Build development tools using the grammar, file bugs/feature requests/etc.

Possible Future Work – consulting@disjunctive.llc

PlusCal tree-sitter grammar

- 1-2 months
- Tree-Sitter can handle multiple languages in a single file

TLA+ language server using the grammar

- 4-6 months
- Bring standardized language extension functionality to many platforms

Full alternative TLA+ parser & interpreter

- 1 year
- Having multiple language implementations is very healthy!