The Journal of Open Source Software

Reiz: Structural Source Code Search at Scale

Batuhan Taskaya[1]

1 Python Core Developer

DOI: DOIunavailable

Software

- Review
- Repository
- Archive

Editor: Pending Editor
Reviewers:

- @Pending Reviewers

Submitted: N/A
Published: N/A

Summary

Reiz is a structural source code search engine that can execute queries for partially known syntactical constructs inside source code. It allows collection and sampling of source code, as well as serialization and comes bundled with a DSL called ReizQL which offers the ability to express fragmentary knowledge about the targeted constructs.

Statement of need

The fact that developers search source code every day is undeniable. This need to search has various reasons by different groups of people. When introducing new features to a language, developers often need to see what kind of an impact that those will have before actually bothering to implement it (or even discuss it in the first place). Prior to making any changes on a publicly facing API, maintainers of those libraries do the pre-requisite work of collecting samples and estimating the ramifications that operation might cause. When the documentation of a framework doesn't sustain the curiosity, searching for a structure (e.g a function, a constant) to see how it can be utilized in real-world software is a common need among developers [Xia et al. (2017)].

While searching for a particular structure inside source code, it is almost impossible to describe structural patterns on a search engine where the code has behaved no different than a stream of characters/tokens. Even on the providers where they support regular expressions, identifying nested syntactical structures or leaving room for some ambiguity is quite problematic.

State of the field

Popular source code search engines (e.g Github Code Search [*Github Code Search* (2021)]) uses full-text search where the code is behaved not much different than a regular textual document. Even though this approach works for some basic queries, structurally it can't go further than matching token sequences. This often causes seeing irrelevant search results on complex queries, or even not being able to express the search itself in a purely textual form. In the past, there has been some work done regarding making queries more expressive through regular expressions (one example might codesearch.debian.net [Stapelberg (2012)]), and even annotating the result set with some semantical and structural knowledge (via finding and resolving API names [Bajracharya et al. (2014)]).

The Journal of Open Source Software

Method

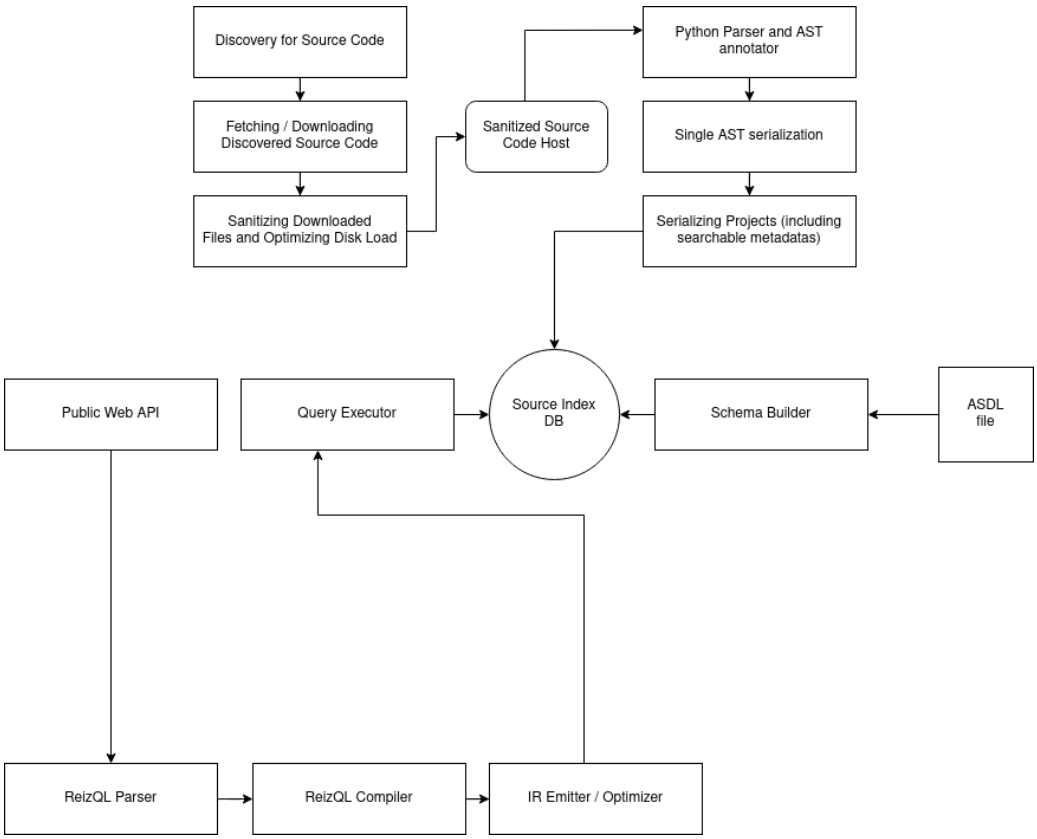

Figure 1: Stages from the Reiz's pipeline.

The internals consists of a pipeline that enables the ability to plug in and out different components, such as for frontends of different languages. The primary piece that every other component directly or indirectly interacts with is the Index DB (a.k.a source warehouse) where the serialized AASTs (Annotated Abstract Syntax Trees) are being held. It is based on an EdgeDB ["EdgeDB" (2020)] instance which interprets the compiled queries and returns the raw result set. The schema used in the Index DB is in the format of ESDL (EdgeDB Schema Definition Language) and automatically generated from the host language's ASDL [Wang et al. (1997)] declaration. It is a common format used by many different projects, most notably by CPython itself.

Sampling Source Code

The source code sampling starts with the collection of the most used python packages, according to the download statistics over the PyPI [Kemenade & Si (2021)]. The list then gets cross-linked to the project's corresponding source control platforms (so that, we can reference the revision that we are fetching). Later on, the data gets downloaded via `git` and then gets sanitized until there is nothing left besides valid source files for the host language.

Subsequently, files get parsed to the AST form offered by the host language, and then annotated with some static knowledge, so that the computation of these properties won't cost anything on the runtime. The annotations include node tags (a unique identifier for a piece of AST that will be the same every time the same structure is annotated, like tree hash), ancestral information (like a set of 2-element tuples, where the first one points to the parent type

and the the second one points to the field that the child belongs to) and metadata regarding the project (like the filename, project name, GitHub url). Afterward the annotated AST gets serialized into the Index DB.

Query Compiler

```
start                  ::= match_pattern

pattern                ::= negate_pattern
                         | or_pattern
                         | and_pattern
                         | match_pattern
                         | sequential_pattern
                         | reference_pattern
                         | match_string_pattern
                         | atom_pattern

negate_pattern         ::= "not" pattern
or_pattern             ::= pattern "|" pattern
and_pattern            ::= pattern "&" pattern
match_pattern          ::= NAME "(" ",".argument+ ")"
sequential_pattern     ::= "[" ",".(pattern | "*" IGNORE)+ "]"
reference_pattern      ::= "~" NAME
atom_pattern           ::= NONE
                         | STRING
                         | NUMBER
                         | IGNORE
                         | "f" STRING

argument               ::= pattern
                         | NAME "=" pattern

NONE                   ::= "None"
IGNORE                 ::= "..."
NAME                   ::= "a".."Z"
NUMBER                 ::= INTEGER | FLOAT
```

ReizQL is a declarative pattern matching language designed specifically for ASTs. It offers an extensive ability to match both full and partial syntax trees and retrieve the results as raw source code. Besides matching trees, it also allows a limited metadata search (filenames, project names etc) and finding alike strings.

Evaluation

For the limited subset of things that can be described in any of the competitor engines, we evaluate the performance of `Reiz` by running similiar queries in Github Code Search [*Github Code Search* (2021)] grep.app [*Grep.app* (2021)] and Krugle [*Krugle.com* (2021)] and report back the amount of true / false positives.

Comparisons

The method of evaluation is running the queries and analyzing the top 10 results. Some engines offer multiple matches per result, and they will be marked as true positive if any of them are a true positive. We also discard match spans, since none of the competitors can successfully display the expression boundaries.

Objective: search for a `len(...)` call

engine	query	true positives	false positives
Github Code Search	`language:python len()`	5	5
grep.app	`len\((.*)\)`	10	0
Krugle (advanced)	`len functioncall:len`	10	0
Reiz	`Call(Name("len"))`	10	0

Objective: search for an addition or a subtraction operation

engine	query	true positives	false positives
Github Code Search	`language:python + -`	0	0
Krugle (fuzzy)	`expr + expr / expr - expr`	0	0
Krugle (solr syntax)	`\+ \-`	2	8
Krugle (regex syntax)	`(.*)(\+|\-)(.*)`	1	9
grep.app	`(.*)(\+|\-)(.*)`	2	8
Reiz	`BinOp(op=Add() | Sub())`	10	0

Objective: search for a return statement that returns a tuple `return ..., ...`

engine	query	true positives	false positives
Github Code Search	`language:python return ,`	2	8
Krugle (solr syntax)	`return \,`	1	9
Krugle (regex syntax)	`return ((.*)),\s*(.*))+`	0	10
grep.app	`return ((.*)),\s*(.*))+`	0	10
grep.app (2)	`return \(((.*)),\s*(.*))+\)`	9	1
Reiz	`Return(Tuple())`	10	0

Bajracharya, S., Ossher, J., & Lopes, C. (2014). Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming*, *79*, 241–259. https://doi.org/10.1016/j.scico.2012.04.008

EdgeDB: The next generation relational database. (2020). In *GitHub repository*. EdgeDB. https://github.com/edgedb/edgedb

Github code search. (2021). GitHub. https://grep.app

Grep.app. (2021). grep.app. https://grep.app

Kemenade, H. van, & Si, R. (2021). *Hugovk/top-pypi-packages: Release 2021.04*. https://doi.org/10.5281/zenodo.4657163

Krugle.com. (2021). krugle.com. https://krugle.com

Stapelberg, M. (2012). Debian code search. In *Bachelor-Thesis*. Hochschule Mannheim Fakultät für Informatik. http://codesearch.debian.net/research/bsc-thesis.pdf

Wang, D. C., Appel, A. W., Korn, J. L., & Serra, C. S. (1997). The zephyr abstract syntax description language. *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, 17.

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., & Xing, Z. (2017). What do developers search for on the web? *Empirical Softw. Engg.*, *22*(6), 3149–3185. https://doi.org/10.1007/s10664-017-9514-4