The Journal of Open Source Software

Reiz: Structural Source Code Search

Batuhan Taskaya[1]

1 Python Core Developer

DOI: 10.21105/joss.03296

Software

- Review
- Repository
- Archive

Editor: Matthew Sottile

Reviewers:

- @lutzhamel
- @yuhc

Submitted: 24 April 2021
Published: 24 June 2021

Summary

Reiz is a structural source code search engine that can execute queries written in ReizQL to retrieve partially known syntactical constructs inside a pre-processed source index.

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)].

For the problems mentioned above, we present Reiz: a source code search engine backend that can exercise queries to match syntax trees in order to leverage the existing language syntax to describe partial knowledge (e.g a searching for a `try`/ `finally` construct without knowing what is under the `try` block).

State of the field

Popular source code search engines (e.g Github Code Search [*Github Code Search* (2021)]) use full-text search where the code is treated no differently 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 semantic and structural knowledge (via finding and resolving API names [Bajracharya et al. (2014)]). There also have been various tools ["Fast" (2017)], ["Astsearch" (2014)] to search AST patterns within source code, but with a limited query format and in a file-by-file basis (no index database) which makes them quite hard to work with in order to examine a large dataset of source code.

Method

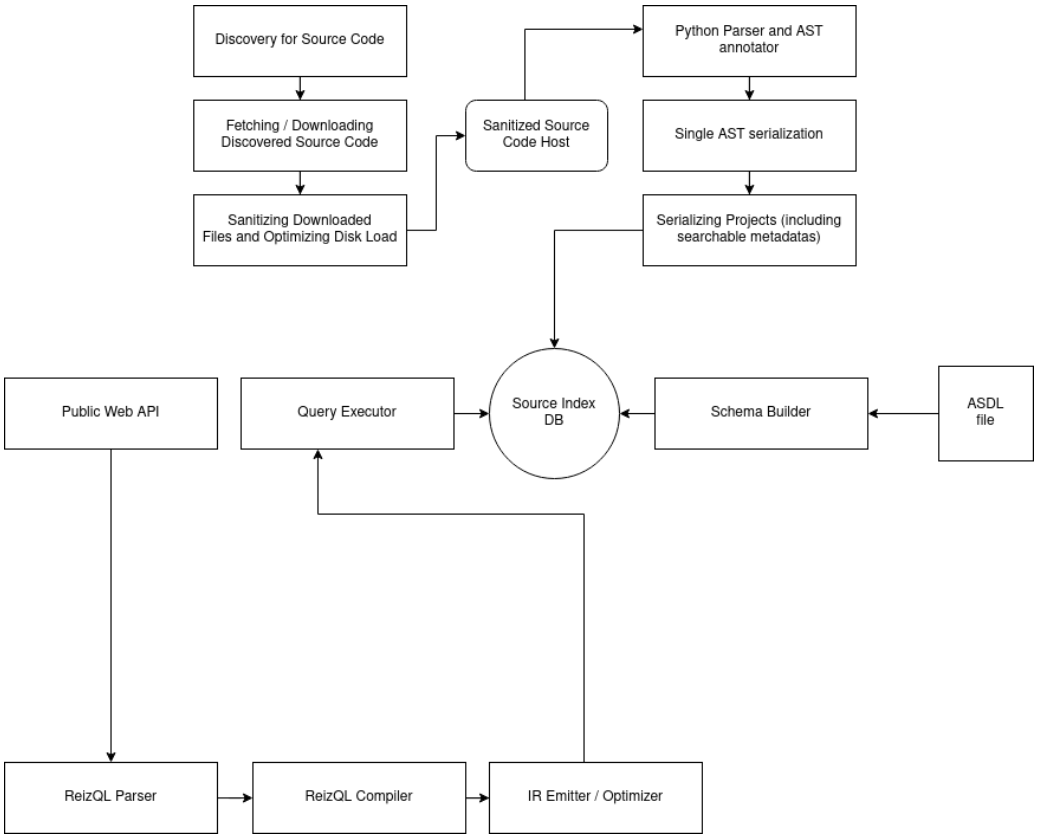

Figure 1: Stages from the Reiz's pipeline.

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. ASDL is a common format used by many different projects, most notably by CPython itself.

Sampling Source Code

Reiz is able to sample source code from a list of projects on a git remote (e.g GitHub). The reference implementation for Reiz comes with an indexer which can automatically construct a list for the most popular packages on PyPI, according to the download statistics [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 at 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
```

The grammar above describes the ReizQL language which is embedded into the execution engine. It can be used as is, or can be selected as a compilation target for a higher level language that is more integrated with the syntax of the host language.

Following query will search for all occurrences of a for loop, where the target's `result` method is called subsequently in the loop body. The `target` is an example of the reference pattern, which has a query-bound value (e.g if the first `~target` capture is `X`, then the following capture is also expected to be the same).

```
For(
    target=~target,
    body=[
        Expr(
            value=Call(
```

```
                func=Attribute(value=~target, attr="result")
            )
        )
    ],
)
```

The query in the example can be automatically generated through various forms, as well as being hand written. For example, the IRun project targets ReizQL with a python superset form to be a more human friendly interface to the engine:

```
for $target in ...:
    $target.result()
```

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. Each result that contains an exact match with the intended objective, e.g a call to `len(...)` function will be counted as a true positive, and otherwise will be counted as a false positive. Some engines offer multiple matches per result, and they will be marked as true positive if any of them checks out with the objective. 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

engine	query	true positives	false positives
grep.app	`return ((.*))(,\s*(.*))+`	0	10
grep.app (2)	`return \(((.*))(,\s*(.*))+\)`	9	1
Reiz	`Return(Tuple())`	10	0

Conclusion

On all three objectives, Reiz got all the matches as true-positive due to due to it's ability to leverage syntax tree structure as well as other annotations that it could collect at pre-processing stage (such as node boundaries to report the exact location) unlike others where the source code is exercised like a regular textual document.

References

Astsearch: Intelligently search in python code. (2014). In *GitHub repository*. takluyver. https://github.com/takluyver/astsearch

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

Fast: Find in AST. (2017). In *GitHub repository*. jonatas. https://github.com/jonatas/fast

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