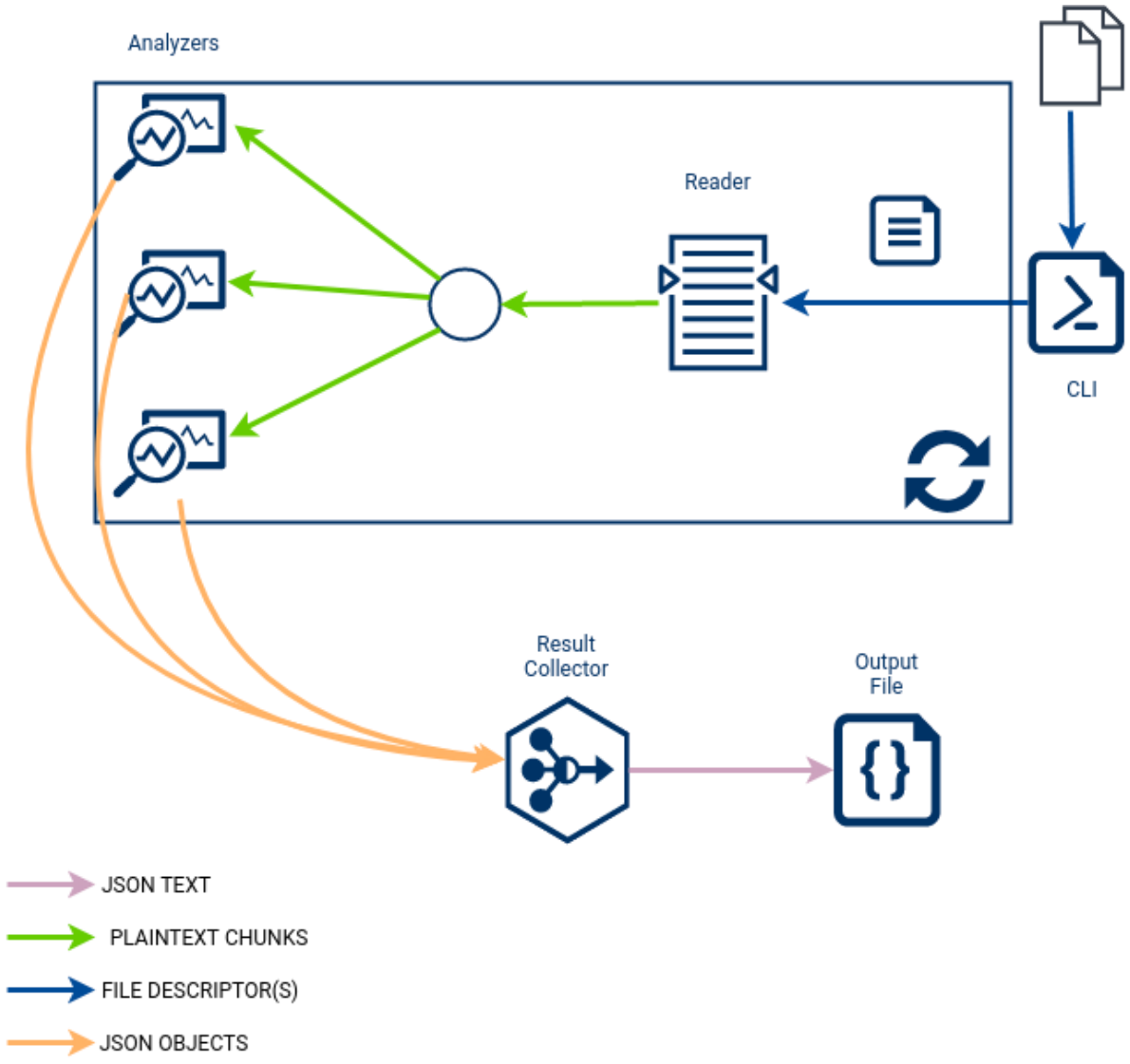
PA193 Semestral project

Ádám Ruman, Andrej Tomči, Roman Chrenšť

Design



Design principles:

The program supports synchronous work only. The files are processed one by one, by iteratively reading it in fixed size chunks. Each chunk is then passed to each of the analyzers, which are objects that process the chunks and update their states. After the whole file has been processed, the analyzers are “asked” to translate their state into JSON objects, which can be written into the output file.

The program is set up for “soft” error handling, so if a failure emerges while processing a specific input, this does not kill the whole program, rather the user is informed, and we move on to the next file.

Support tools

Github

We have prepared a Github repository with Actions set up for running build, tests, and scoring system.

Build & tests

Build and tests are run on every push to *devel* and *main* branch and pull requests to *main*. This workflow tests that the code can be compiled, and thanks to the properties of a rather strict rust compiler, it also gave us insights into security and other issues.

Scoring

Scoring works thanks to the provided dataset and python script. We prepared a Github workflow that triggers on every pull request and automatically comments the pull request with the current average score (calculated from all input files in the dataset). This allowed us to incrementally improve our parser and to easily check that we are moving towards the goal (average score as high as possible).

Used analysis tools

Lint – Cargo Clippy

We used a static analyzer in our Github actions to continuously check our code.

The rustc compiler

Rust's compiler itself is a great static analysis tool!

Valgrind

When we finalized our implementation, we checked the program using valgrind. The output showed us that some memory was possibly lost but it was tied to the use of regexes.

Radamsa fuzzer

We used the radamsa fuzzer to test how the program handles files with various content, mainly focusing on the corrupted files. Thanks to it we found two errors and fixed them.

Microsoft RegexFuzz

We fuzzed some of our regexes with this tool, using the utf-8 charset. Some regexes could not be fuzzed as the tool does not support nested quantifiers and some of the rust-regex syntax.

GitGuardian

This tool was run to ensure no confident data are leaked in our GitHub repository.

Running the tool

If you encounter any issue with running the tool, feel free to contact us!

1. Install rust - https://doc.rust-lang.org/book/ch01-01-installation.html
2. Clone project – $ git clone https://github.com/addam128/cextractor
3. Change directory - $ cd cextractor

4. Run the tool - $ cargo run --release dataset/file.txt dataset/file2.txt …