

Introduction

The artefact provides relevant samples of code from each section in the paper, illustrating the behaviour of the ownership and borrowing system we develop. These are bundled with a Docker image containing a Granule interpreter augmented with the extensions discussed in the paper. These both illustrate the general principles of ownership and borrowing captured by our system and also demonstrate the interfaces for mutable arrays and polymorphic references which are described in later sections of the work.

The artefact also includes the Rust code snippets from Section 2, in the `rust` folder. The Rust compiler itself is not provided, as these are not an essential part of the artefact, but they are provided for completeness and to allow for free experimentation with any standard installation of Rust.

Hardware Dependencies

There are no special hardware or software requirements for the artefact beyond the standard system requirements for Docker and Granule. Around 80MB of free space will be needed to store the uncompressed Docker image.

Note that the Docker image is built on an x86-64 (AMD64) system, so if the user is running the system on an M1/M2 Mac then it may be necessary to install Rosetta (eg by running `sudo softwareupdate --install-rosetta --agree-to-license`) and enable the “Use Rosetta for x86/amd64 emulation on Apple Silicon” option in the Docker settings menu. The image has been tested on such a system, so we can confirm that running the artefact through Rosetta successfully is possible, though considerably slower due to the emulation.

If the user wishes to experiment with the code on their own system, rather than through the Docker image, then this should also be feasible. Granule binaries and instructions for building the Granule compiler from source can be found at <https://github.com/granule-project/granule>; the version required to compile and run the examples presented here can be found on the `fractional` branch, with the latest version at time of writing labelled by the following tag: <https://github.com/granule-project/granule/tree/OOPSLA2024-artefact>

Getting Started Guide

Load the file provided via:

```
docker load < artefact.tar.gz
```

This may take some time as Docker will unpack the zip file before loading the image. The Docker image can now be started as with any other Docker image; first run `docker images` to find the image ID. For those less familiar with Docker, look for the row with repository `ghcr.io/raehik/granule-repl`, then

the image ID is in the third column and has 12 characters. Next, we recommend using the below command to start an interactive Granule REPL session, with the local directory (containing the example code files) mounted as `/host`.

```
docker run -v $(pwd):/host -it <IMAGE-ID>
```

The reviewer will now be loaded into Granule’s interactive environment. Here, the command `:l host/examples.gr` (for instance) will load in the file `examples.gr`, assuming that the image was run from the directory containing the artefact. Successfully type-checking this file with an output as shown below (or, indeed, any of the other Granule code samples provided) is sufficient for the reviewer to confirm that the artefact has been installed correctly for the kick-the-tyres stage of artefact evaluation.

```
> docker run -v $(pwd):/host -it <IMAGE-ID>
Welcome to Granule interactive mode (grepl). Version 0.9.5.0
Granule> :l host/examples.gr
host/examples.gr, checked.
```

To quit the Granule REPL, type `:q`.

Step by Step Instructions

The primary purpose of this artefact is for a user to be able to experiment with the examples presented in the paper in a practical setting, in order to aid understanding. There are no experimental results to replicate; the focus is on providing additional demonstrations of the ideas.

Four example Granule files are provided alongside the artefact image, which can all be loaded directly into the Granule REPL or freely modified and edited. These are:

- `examples.gr` - a file containing all of the Granule code snippets presented throughout the paper, with pointers to their locations within the paper; this allows a reviewer to follow along with the paper and make sure the examples type-check.
- `simple-array.gr` - some simple illustrations of the mutable array interface introduced in Section 5 (p.7)
- `simple-ref.gr` - some simple illustrations of the polymorphic reference interface introduced in Section 5 (p.7)
- `parsum.gr` - a more involved example where two halves of a mutable array are borrowed and summed in parallel, in order to demonstrate all the advances of the paper in a more real-world setting.

Once a file has been loaded into the REPL via the `:l` command as described above, values in that file can be evaluated; for example, loading `host/simple-ref.gr`

and evaluating `test` will print 42.0. It is also possible to check the type of a function or definition (including primitives) using `:t`:

```
Granule> :l host/simple-ref.gr
host/simple-ref.gr, checked.
Granule> test
42.0
Granule> :t test
test : Float [6]
Granule> :t swapRef
swapRef : forall {a : Type, f : Fraction, id : Name} . {mut f} =>
  & f (Ref id a) -> a -> (a, & f (Ref id a))
```

The reviewers should feel free to modify these files as they wish, and experiment with what is permissible and what leads to type errors. The modified files can be loaded in using `:l` as above, or the `:r` command can be used to reload the most recently loaded file.

Reusability Guide

The artefact is primarily reusable in the sense that the reviewers are not limited to the example code provided - you are welcome to write your own programs following the ownership and borrowing principles described in the paper, and use the Granule interpreter to check and evaluate them. We invite interesting contributions (and bug reports!) from keen evaluators.

In addition, the source code for both Granule itself and our extension presented here is not only available to the reviewers to examine, but also is fully open source. As mentioned previously, the particular version embedded in this artefact is available at the following tag:

<https://github.com/granule-project/granule/tree/OOPSLA2024-artefact>

Reviewers should feel free to download the code and explore the source tree for themselves in order to better understand how the implementation corresponds to the theoretical system described in the paper. The changes made to the Granule code base for this work can be mostly seen here:

<https://github.com/granule-project/granule/compare/dev-minor...fractional>