Concurrent Data Structures Linked in Time

Germán Andrés Delbianco, Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee

Artifact Description

Thank you for agreeing to review the artifact for our ECOOP 2017 submission entitled *"Concurrent Data Structures Linked in Time"*. The artifact is represented by the Coq sources of the implementation of Fine-grained Separation Logic (FCSL) in the Coq proof assistant, as well as the case studies conducted for the ECOOP'17 submission.

There are two possible ways to test the development: compiling FCSL from the source files or running the provided virtual machine. We provide further instructions in the sequel. A brief commentary with respect to this paper's specific case studies is also provided below. Moreover, the `README.md` file also provides information about previous FCSL developments and case studies.

Finally, if you are interested in finding out more details about the implementation of FCSL and several case studies, we encourage you to check the FCSL reference manual: `http://software.imdea.org/fcsl/papers/fcsl-manual.pdf`

Building FCSL from source

Build FCSL from the Coq sources files packed in the ZIP archive (`fcsl-ecoop17.zip`). In order to compile and check the sources and case studies from the papers, you will need to have Coq 8.5pl3 and Ssreflect 1.6 installed. They can be downloaded from the following sites:

- `https://coq.inria.fr/coq-85`
- `https://math-comp.github.io/math-comp/`

However, we recommend installing both Coq and Ssreflect using the OPAM package manager. With OPAM installed, copy and paste the following commands to build Coq 8.5.3 and Ssreflect, together with their dependencies.

```
opam repo add coq-released https://coq.inria.fr/opam/released

opam pin add coq 8.5.3

opam install coq-mathcomp-ssreflect
```

We also recommend to use Emacs with installed Proof General mode[1] to browse and step through the sources.

Once you have Coq and Ssreflect installed, follow the instructions in the `README.md` file of the `fcsl-ecoop2017.zip` archive to build the project. It takes around 100 minutes to build the project and all the case studies, so we thank you in advance for your patience.

The `README.md` file also provides commentary on the case studies from this paper, which are located in the folder `Examples/Relink`, as well as a description of previous FCSL case studies.

[1] `https://github.com/emacsattic/proofgeneral`

Using the VirtualBox VM image

You can also use the VirtualBox VM image with Coq/Ssreflect/Emacs/Proof General installed. The VM image is available by the link in the artifact description. It requires VirtualBox 5.0[2] to run. Whenever necessary, use `fcsl` as both the username and password.

Once successfully launched, navigate to the `~/fcsl-ecoop17` folder to browse the project sources. The project is already compiled, but you can follow the instructions from `README.md` file in order to re-build the project. This will automatically check all the proofs.

Use Emacs with Proof General mode to step through *.v files from the `Examples/Relink` folder.

A brief summary of this paper's Coq development

Folder: `Examples/Relink`

The folder `Examples/Relink` folder contains the main case study of the *Linking in Time* technique for verifying concurrent data structures with non-fixed linearization points introduced in the paper. It consists of several files implementing the verification in FCSL of Jayanti's single writer/single scanner snapshot construction:

- `jayanti_snapshot.v` — Preliminaries. Lemmas for reasoning about coloring patterns of histories, and other history invariants.
- `jayanti_ghoststate.v` — Implementation of the auxiliary state for the snapshot object, its transitions, invariants and associated invariant preservation proofs.
- `jayanti_concurroid.v` — Implementation of the FCSL concurrent resource —a.k.a. *concurroid* for the snapshot.
- `jayanti_actions.v` — FCSL atomic actions for the snapshot object concurrent resource.
- `jayanti_stability.v` — Assertions used in the proof outline of the main methods and proofs of their stability under FCSL transitions.
- `jayanti_library.v` — Implementation and verification of the atomic commands (auxiliary code) and the snapshot library (i.e. scan and write procedures).
- `jayanti_clients.v` — Verification of the clients of the snapshot data structure, as presented in Section 4 of the paper.

In the sequel, we relate the different parts of the development with their corresponding presentation in the paper:

The invariants described in Section 5, together with the implementation of the concurrent resource for the snapshot object are defined in the `jayanti_ghoststate.v` and `jayanti_concurroid.v` files. In the later file, there is also the implementation of the auxiliary code functions (a.k.a transitions in FCSL jargon) from Section 6. The proof of the 2-state invariants from Invariant 1, and those from Invariant 2, together with the definitions of the main assertions (e.g. the *stable order* Ω) are carried out in `jayanti_stability.v`. The verification of the `write` and `scan` files, as presented in Section 7, are carried our in the file `jayanti_library.v`.

Finally, the clients in Section 4 are implemented and verified in `jayanti_clients.v`. We refer the reader to the descriptions in the `README.md` file of the project and/or the accompanying manuscript for further detail.

[2] `https://www.virtualbox.org/wiki/Downloads`

Folder: `Core`

Two new libraries were added to the Core of existing FCSL libraries: a new library for *relinkable* histories, added to `histories.v` and an extended theory of algebraic surgery operations for lists, which were necessary to prove the properties of the *relink* operation. The latter is implemented in `seq_extras.v`.