Concurrent Data Structures Linked in Time (Artifact)*

Germán Andrés Delbianco[1], **Ilya Sergey**[2], **Aleksandar Nanevski**[3], and **Anindya Banerjee**[4]

1 **IMDEA Software Institute, Madrid, Spain and Universidad Politécnica de Madrid, Spain**
german.delbianco@imdea.org

2 **University College London, United Kingdom**
i.sergey@ucl.ac.uk

3 **IMDEA Software Institute, Madrid, Spain**
aleks.nanevski@imdea.org

4 **IMDEA Software Institute, Madrid, Spain**
anindya.banerjee@imdea.org

Abstract

This artifact provides the full mechanization in FCSL of the developments in the companion paper, "Concurrent Data Structures Linked in Time". In the latter, we propose a new method, based on a separation-style logic, for reasoning about concurrent objects with such linearization points. We embrace the dynamic nature of linearization points, and encode it as part of the data structure's *auxiliary state*, so that it can be dynamically modified in place by auxiliary code, as needed when some appropriate run-time event occurs. We illustrate the method by verifying (mechanically in FCSL) an intricate optimal snapshot algorithm due to Jayanti, as well as some clients.

FCSL is the first completely formalized framework for mechanized verification of full functional correctness of fine-grained concurrent programs. It is implemented as an embedded domain-specific language (DSL) in the dependently-typed language of the Coq proof assistant, and is powerful enough to reason about programming features such as higher-order functions and local thread spawning. By incorporating a uniform concurrency model, based on state-transition systems and partial commutative monoids, FCSL makes it possible to build proofs about concurrent libraries in a thread-local, compositional way, thus facilitating scalability and reuse: libraries are verified just once, and their specifications are used ubiquitously in client-side reasoning.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, D.2.4 Software/Program Verification, F.1.2: Parallelism and concurrency, D.1.3 Concurrent Programming

Keywords and phrases separation logic, linearization points, concurrent snapshots, FCSL

Digital Object Identifier 10.4230/DARTS.3.2.4

Related Article Germán Andrés Delbianco, Ilya Sergey, Aleksandar Nanevski and Anindya Banerjee, "Concurrent Data Structures Linked in Time", in Proceedings of the 31st European Conference on Object-Oriented Programming (ECOOP 2017), LIPIcs, Vol. 74, pp. 8:1–8:30, 2017.
http://dx.doi.org/10.4230/LIPIcs.ECOOP.2017.8

Related Conference European Conference on Object-Oriented Programming (ECOOP 2017), June 18-23, 2017, Barcelona, Spain

[*] This research is partially supported by EPSRC grant EP/P009271/1, the ERC consolidator grant Mathador–DLV–724464, and the US National Science Foundation (NSF). Any opinion, findings, and conclusions or recommendations expressed in the material are those of the authors and do not necessarily reflect the views of NSF.

Dagstuhl Artifacts Series, Vol. 3, Issue 2, Artifact No. 4, pp. 4:1–4:4
Dagstuhl Artifacts Series
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Scope

The scope of this artifact is to provide the full mechanization of the developments described in the companion paper, "Concurrent Data Structures Linked in Time". The mechanization consists in the development of the infrastructure to implement the *"linking in Time"* technique as an FCSL library, the implementation of Jayanti's snapshot object in FCSL, together with the verification of the library methods and the clients presented in the paper. Moreover, for completeness sake, we have bundled this new development together with previous case studies in FCSL. Finally, if you are interested in finding out more details about the implementation of FCSL [3, 2] and several previous case studies [5, 4, 6], we encourage you to check the FCSL reference manual [1].

2 Content

This artifact extends the previous release of FCSL with the Coq source files mechanizing the proofs of the developments presented in the associated research paper. For the sake of brevity, we list only the new contributions, and refer the reader to the FCSL user manual for details about previous case studies and the development of FCSL.

Folder: `Examples/Relink` The folder `Examples/Relink` 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 files `jayanti_ghoststate.v` and `jayanti_concurroid.v` define the invariants described in Section 5, together with the implementation of the concurrent resource for the snapshot object. Moreover, 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 the FCSL manual for further detail.

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`

3 Getting the artifact

The artifact endorsed by the Artifact Evaluation Committee is available free of charge on the Dagstuhl Research Online Publication Server (DROPS). The latest version of this artifact is available also on the FCSL project website: `http://software.imdea.org/fcsl`.

4 Tested platforms

The artifact is known to work on any platform running Oracle VirtualBox version 5[1] with at least 5 GB of free space on disk and at least 4 GB of free space in RAM. Alternatively, FCSL can be built from the sources available from the project's webpage. The latter can be done in any system capable of installing and executing Coq v.8.5pl3[2] and Ssreflect 1.6.[3]

5 License

GPLv3 (`https://www.gnu.org/licenses/gpl`)

6 MD5 sum of the artifact

95b0fc97f64958028ef3d1b31a4ffa4e

7 Size of the artifact

2.75 GB

References

1 Aleksandar Nanevski, Ruy Ley-Wild, Ilya Sergey, , Anindya Banerjee, and Germán Andrés Delbianco. FCSL: Fine-grained concurrent separation logic. `http://software.imdea.org/fcsl/`.

2 Aleksandar Nanevski, Ruy Ley-Wild, Ilya Sergey, Anindya Banerjee, and Germán Andrés Delbianco. Mechanized verification of fine-grained concurrent programs in fine-grained concurrent separation logic: User manual and code commentary. `http://software.imdea.org/fcsl/papers/fcsl-manual.pdf`.

3 Aleksandar Nanevski, Ruy Ley-Wild, Ilya Sergey, and Germán Andrés Delbianco. Communicating state transition systems for fine-grained concurrent resources. In Zhong Shao, editor, *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014. Proceedings*, volume 8410 of *LNCS*, pages 290–310. Springer, 2014. `doi:10.1007/978-3-642-54833-8_16`.

4 Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. Mechanized verification of fine-grained concurrent programs. In David Grove and Steve Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2015*, pages 77–87. ACM, 2015. `doi:10.1145/2737924.2737964`.

5 Ilya Sergey, Aleksandar Nanevski, and Anindya Banerjee. Specifying and verifying concurrent algorithms with histories and subjectivity. In Jan Vitek, editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015. Proceedings*, volume 9032 of *LNCS*, pages 333–358. Springer, 2015. `doi:10.1007/978-3-662-46669-8_14`.

[1] `https://www.virtualbox.org/`
[2] `https://coq.inria.fr/coq-85`
[3] `https://math-comp.github.io/math-comp/`

6 Ilya Sergey, Aleksandar Nanevski, Anindya Banerjee, and Germán Andrés Delbianco. Hoare-style specifications as correctness conditions for non-linearizable concurrent objects. In Eelco Visser and Yannis Smaragdakis, editors, *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016*, pages 92–110. ACM, 2016. doi:10.1145/2983990.2983999.