On Algebraic Abstractions for Concurrent Separation Logics

This artefact is a companion to the paper *On Algebraic Abstractions for Concurrent Separation Logics*. The artefact contains `Coq` sources of the developments presented in the submission. The artefact supports the developments paper in both a theoretical and a practical way.

First, it provides a complete bottom-up mechanization of partial commutative monoids (PCM), separating relations, PCM morphisms, and the related constructions. The artefact formalizes all the concepts defined in the paper.

Secondly, the artifact demonstrate practical utilisation of the theory of PCMs. Using `fcsl` (Nanevski et al, 2019) as the opaque type theory, the artifact provides mechanical verification of Ticket lock, the running example developed in the paper. The artefact also contains additional examples that the main body submission does not discuss (see Additional Artefact Description for details).

Contents

- List of Claims
  - Section 2 - PCM Abstractions by Example
  - Section 3 - PCM Abstractions formally
  - Section 4 - Invertible morphisms and separating relations
- Download, Installation, and Sanity-Testing
  - Booting Up
  - Sanity-Testing
- Evaluation Instructions
  - Type-Checking, Proof-Checking and Consistency
  - Evaluating-the-Claims-of-the-Paper
  - Reusability of the Theory of PCM
- Additional Artefact Description
  - Structure of the Project
  - The Virtual Machine Image

List of Claims

The paper *On Algebraic Abstractions for Concurrent Separation Logics* makes the following claims in order they occur in the paper:

Section 2 - PCM Abstractions by Example

The Ticket lock example is developed in the file `examples/ticketlock.v`.

Section 2.5

The programs `lock` and `unlock` with ghost code annotations and concrete specifications are program definitions `lock` and `unlock` (on lines 488 and 515 respectively). In comparison to the paper, these specs are carried out more generically, in large-footprint style, i.e. for an arbitrary history `h` rather then for the empty history `\emptyset`.

Section 2.6

The morphism `\alpha` is defined as definition `alpha` (line 558) and its separating relations as `alpha_rel` (line 539).

The abstract specs are given as program definitions `lock_alpha` and `unlock_alpha` (lines 616 and 626 respectively).

Section 2.7

The specifications utilising sub-PCM construction and the morphisms `alpha'` (defined on line 663) are given as `lock'` and `unlock'` (lines 666 and 676 respectively).

Section 3 - PCM Abstractions formally

Section 3.1

Definition 3.1 - TPCMs are defined in the file `pcm/pcm.v` (module `TPCM`). In comparison to the paper, the definition in the formalisation is stratified into plain PCMs and PCMs that possess top (`T`) element (property 4 of Def. 3.1). Note that we implement algebraic structures as packed classes thus the definition of TPCMs spans several Coq definitions in the module; similarly for other algebraic structures in this list.

Proposition 3.3 - the proposition is given by the canonical structure `prodTPCM` in `pcm/pcm.v` (line 633).

Section 3.2

Definition 3.4 - separating relations are defined in the file `pcm/morphisms.v` (module `SepRel`). In comparison to the paper, separating relation are defined more general and do not impose *strengthening* (property (2) of Def 3.4) and as a consequence, the assumption (`x \bot y` , denoted `valid (x \+ y`) in the code) is carried around explicitly where needed.

The actual axioms of Def 3.4 are given as `orth_axiom` and the equivalence with the definition `seprel_axiom` in the presence of the assumption `valid (x \+ y)` is given by lemma `orth_sep` (line 60 of `pcm/morphism.v`).

Proposition 3.5 - `sepU0` in `pcm/morphism.v` (line 260)

Basic examples of separating relations are formalised as `sep0` and `sepT` in `pcm/morphism.v` (lines 286 and 275 respectively).

Separating relation `\bot_alpha` is formalised as `alpha_rel` in `examples/ticketlock.v` (line 539).

Separating relation `ordered` is formalised as `omega_rel` and proven to be a separating relation as lemma `omega_seprel` in `examples/ticketlock.v` (lines 109 and 113).

Non-example of `no_gaps` is defined as `upsilon_rel` and proven not to be a separating relation (`upsilon_non_seprel`) in `examples/ticketlock.v` (lines 139 and 141).

Section 3.3

Definition 3.6 - structure `morphism` in `pcm/morphism.v` (line 408).

The basic examples are formalised as canonical structure `id_morph` (file `pcm/morphism.v` , line 704) and `fst_morph` and `snd_morph` (file `pcm/morphims.v` , lines 754 and 759).

Definition 3.7 - the definitions are given in the file `pcm/morphism.v` , composition is given by `comp_morph` (line 712), tensor product as `pmorph_morph` (line 908), and arrow product as `fprod_morph` (line 743).

Theorem 3.8 - the categorical properties of morphisms are show in the file `pcm/morphisms.v` in section `CategoricalLaws` (line 719).

Definition 3.9 - kernel and equalizer are formalised in file `pcm/morphism.v` as `ker` and `eqlz` (lines 500 and 509 respectively) and shown to be separating relations as canonical structures `ker_seprel` (line 624) and `eqlz_seprel` (line 655) in the same file.

Definition 3.10 - restriction is formalised in file `pcm/morphism.v` as `res` (line 505) and shown to be a morphism by the canonical structure `res_morph` (line 634).

Example 3.11 - that a filter over a finite map is a morphisms is shown in file `pcm/unionmap.v` as the canonical structure `umfilt_morph` (line 3473). The size function is provided by `ssreflect` library and we do not show it is a morphism.

Section 3.4

Definition 3.12 - Sub-PCMs are defined in file `pcm/morphism.v` in module `SubPCM` (line 926), the injection `\iota` is formalised as `pval` and the retraction `\rho` as `psub`.

Concrete construction of a sub-PCM is given in file `pcm/morphism.v` in module `SepSubPCM` (line 1090), in particular:

Lemma 3.13 - the properties are show in file `pcm/morphisms.v` in order as lemmata `xsep_joinC` (line 1129), `xsep_joinA` (line 1152), `xsep_unitL` (line 1155), `xsep_valid_undef` (line 1193), `xsep_valid_unit` (line 1169), and `xsep_validL` (line 1162, the right part follows from commutativity).

That the construction gives rise to a sub-PCM - the canonical structure `xsepSubPCM` in `pcm/morphism.v` (line 1258)

Section 4 - Invertible Morphisms and Separating Relations

Section 4.3

Definition 4.1 - invertible separating relations are defined as `inv_rel` in file `pcm/invertible.v` (line 32).

Definition 4.2 - invertible morphisms are defined as `inv_morph` in file `pcm/invertible.v` (line 44).

Proposition 4.3 - lemma `inv_sepT` in file `pcm/invertible.v` (line 49)

Proposition 4.4 - lemmata `inv_comp` and `inv_cprod` in file `pcm/invertible.v` (lines 55 and 71 respectively)

Proposition 4.5 - lemmata `inv_ker` and `inv_eqlz` in file `pcm/invertible.v` (lines 66 and 79 respectively)

Theorem 4.6 - the theorem is generalised from the concrete construction in the paper (i.e. `xsepSubPCM`) to an arbitrary sub-PCM over the appropriate separating relation as `inv_comp_sub` in file `pcm/invertible.v` (line 99).

Lemma 4.7 - lemma `inv_alpha_seprel` in file `examples/ticketlock.v` (line 688)

Lemma 4.8 - lemma `inv_alpha` in file `examples/ticketlock.v` (line 695)

Corollary 4.9 - corollary `inv_alpha'` in file `examples/ticketlock.v` (line 704)

Section 4.4

Lemma 4.10 - lemma `duplicable` in file `examples/CAP.v` (line 15).

Lemma 4.11 - lemma `homomorphic` in file `examples/CAP.v` (line 30).

Download, Installation, and Sanity-Testing

We provide a VM image in Open Virtual Appliance (OVA) format, `FCSL-popl2021.ova`, via Zenodo. The provided image can evaluate the FCSL developments out of the box. Any virtualisation software compatible with the OVA format should suffice to work with the provided image. For further details see Additional artefact description.

Booting Up

Obtain the VM image from Zenodo, import the provided VM image into your virtualisation software (supporting the `.ova` standard) and boot it up. It logs in automatically as the `fcsl` user.

Sanity-Testing

The source code of the project is located in `~fcsl/fcsl-popl2021`. The source code has already been compiled to allow immediately inspect it using Emacs/ProofGeneral or CoqIde. As a sanity check, we suggest removing compilation artefacts and rebuilding the project by invoking make from the project directory:

```
cd ~fcsl/fcsl-popl21
make clean && make
```

The build process usually takes from 5 to 10 minutes. The output of the `make` should be the following:

```
fcsl@xubuntu:~/fcsl-popl21 $ make clean && make
coq_makefile -f _CoqProject -o CoqMakefile
make --no-print-directory -f CoqMakefile clean
CLEAN
coq_makefile -f _CoqProject -o CoqMakefile
make --no-print-directory -f CoqMakefile
COQDEP VFILES
COQC options.v
COQC core/axioms.v
COQC core/prelude.v
COQC core/pred.v

...  (31 lines omitted)
```

Note that the build process output does not contain neither errors nor warnings when running the versions of `Coq` and `Mathcomp` library provided.

Evaluation Instructions

CoqIDE and Emacs (with ProofGeneral) can be used to inspect and navigate the source files in the `~/fcsl-popl2021` folder. Both can be invoked from the terminal, or, alternatively, using launchers on the desktop.

Type-Checking, Proof-Checking and Consistency

Type Checking is Proof Checking. Since we are using a dependently typed theory to mechanize our development, proof scripts are checked by Coq's compiler `coqc`. All the claims in the paper reduce to asserting that a certain definition has a valid Coq Type. This constitutes the sanity check `make clean && make` we described in the previous section.

No Admitted Proofs. Our development does not contain `Admitted` proofs. All the developments presented in the paper are completely mechanized, without unfinished proofs, or hidden meta-theory assumptions. Easy way to observe this is to run the following command from the project directory:

```
find -name '*.v' | xargs grep 'Admitted\.' | wc -l
```

The command counts the admitted proofs:

```
fcsl@xubuntu:~/fcsl-popl21$ find -name '*.v' | xargs grep 'Admitted\.' | wc -l
0
```

On the other hand, the number of closed proofs (there are no `Defined.` proofs) can be computed by the command

```
find -name '*.v' | xargs grep 'Qed\.' | wc -l
```

```
fcsl@xubuntu:~/fcsl-popl21$ find -name '*.v' | xargs grep 'Qed\.' | wc -l
3044
```

Axioms. We do rely though, on two *standard* axioms on top of Coq: propositional and functional extensionality, which are known to be compatible with Coq's underlying type-theory. These are introduced in `core/axioms.v`.

Tactics. Our approach to formalisation requires no particular tactic support when applied in Coq. The way to avoid tactics is by building appropriate mathematical structures and their associated theories. Then the proof development can be done by lemmas, not tactics. In the case of concurrent separation logic, it requires PCM morphisms and separating relations from this paper.

Evaluating the Claims of the Paper

The theory of PCM morphisms is provided in directory `pcm/` while the example is in directory `examples/`. The evaluation is best carried out by comparing the claims of the paper (listed in section List of Claims) and the respective definitions in the code and checking that Coq definition, lemma, etc. corresponds to the statement in the paper. This can be done using CoqIDE or Emacs/ProofGeneral which further allows to step trough the proofs and compare the mechanical proofs with proofs in the paper. Alternatively, the source code can be explored through generated HTML documentation that is available in directory `html/`. This documentation has the benefit of hyperlinks between identifiers and their places of definition, which facilitates understanding of the formal Coq statements. The relevant files are:

- `html/fcsl.examples.ticketlock.html` for the ticket lock example,
- `html/fcsl.examples.CAP.html` for the comparison to CAP,
- `html/fcsl.pcm.pcm.html` for PCMs,
- `html/fcsl.pcm.morphism.html` for morphisms and separating relations, and
- `html/fcsl.pcm.invertible.html` for invertibility.

Note: If you cleaned and rebuilt the project in sanity test step, the HTML documentation was cleaned as well. You might need to rebuild it by running `make html`.

Reusability of the Theory of PCM

The contributions of the paper *On Algebraic Abstractions for Concurrent Separation Logics* is the use of the theory of PCMs for assertion logic. To this effect, we have made the source code available via GitHUB and as an OPAM package coq-fcsl-pcm.

For the ease of evaluation of claims made in the paper we make the theory of PCM, the opaque logic FCSL, and the examples part of one source tree. This choice, as opposed to, e.g., properly making the source code of examples dependent on packages for PCMs and FCSL, allows seamless navigation of source code from within CoqIDE or Emacs.

Additional Artefact Description

Structure of the Project

The structure of the project is desrcribed in a separate file STRUCTURE.md. The file gives details of particular file in this artefact and list addition examples that are provided including their short descripions.

The Virtual Machine Image

The virtual machine image we provide was created using Oracle Virtual Box Version 6.1.14 r140239 (with the corresponding Extension Pack). Both can be downloaded free of charge from https://www.virtualbox.org/wiki/Downloads, and GNU/Linux, macOS, and Windows versions are supported.

FCSL-popl2021.ova

OVA Contents The virtual machine comes with the following components pre-installed:

- Xubuntu 20.04.1 LTS
- OPAM 2.0.5
- OCAML 4.11.1
- Coq 8.12
- Mathcomp 1.11.0 (SSReflect's Mathematical Components Library),
- GNU Emacs 26.3 + ProofGeneral (MELPA: 20200911.3),
- CoqIDE 8.12

The VM is more-over set up with 4GB of RAM, and a 50GB (extensible) virtual hard drive. This settings can be changed using the VM manager to optimize evaluation, but we suggest not reducing the virtualized RAM setting.

Credentials The `fcsl` user's password is `fcsl`, the `sudo` command is enabled for the user `fcsl` (using the same password).

References

Aleksandar Nanevski, Anindya Banerjee, Germán Andrés Delbianco, and Ignacio Fábregas. 2019. Specifying concurrent programs in separation logic: morphisms and simulations. Proc. ACM Program. Lang. 3, OOPSLA, Article 161 (October 2019), 30 pages. DOI:https://doi.org/10.1145/3360587