Overview of the Structure of the Coq Libraries for Zooid

David Castro (Imperial College London) Francisco Ferreira (Imperial College London)
Lorenzo Gheri (Imperial College London) Nobuko Yoshida (Imperial College London)

03/03/2021

Here we give a brief overview on the structure of the Coq libraries, in the artifact for the paper "Zooid: a DSL for Certified Multiparty Computation"; these can be found in the folder `theories`. Figure 1 shows the dependency graphs of the libraries; we follow the graph on the left, which gives a structure of the main development. More detail can be found in the paper and its appendix, with pointers to the code for definitions and theorems.

The file `Common.v` calls the theories in the folder `Common`.

- `Forall.v` and `Member.v` provide support for lists, which will be used for continuations (branching) in inductively defined global and local types.
- In `AtomSets.v` *payload types* for sent/received messages in Zooid are defined.
- `Alt.v` provides support for partial functions, as we use these for continuations (branching) in coinductively defined global and local trees.
- In `Actions.v` the *actions* of our LTS semantics are defined, together with the codatatype of *execution traces*.
- *Queue environments* for asynchronous semantics are defined in `Queue.v`.

The file `Global.v` calls the theories in the folder `Global`.

- In `Syntax.v` the dadatype of *global types* is defined; some basic infrastructure is given.
- In `Tree.v` the codadatype of *global trees* is defined; some basic infrastructure is given.
- `Unravel.v` gives the *unravelling relation*, that pairs global types with their representation as trees.
- In `Semantics.v` we give the semantics for global types: we define a *labelled transition system* for global trees (unravelling of global types) and we introduce the notion of *execution trace*.

The file `Local.v` calls the theories in the folder `Local`.

- In `Syntax.v` the dadatype of *local types* is defined; some basic infrastructure is given.
- In `Tree.v` the codadatype of *local trees* is defined; some basic infrastructure is given.
- `Unravel.v` gives the *unravelling relation*, that pairs local types with their representation as trees.
- In `Semantics.v` we give the semantics for local types: we define a *labelled transition system* for local trees (unravelling of local types) and we introduce the notion of *execution trace*.

The file `Projection.v` calls the theories in the folder `Projection`.

- In `IProject.v` the inductive *projection of a global type onto a participant*, is defined as a recursive function, outputting a local type.

- In `CProject.v` the coinductive *projection of a global tree onto a participant*, is defined as a relation, between a global tree and a local tree.
- `Correctness.v` contains the theory, leading to theorem `ic_proj`, namely "unravelling preserves projection".
- In `QProject.v` Defines the *projection of a global tree on queue environments*, which is fundamental for the following results, connecting the asynchronous LTS semantics of global and local trees.

The file `TraceEquiv.v` concludes the metatheory of multiparty session types. It contains *soundness and completeness* results(`Project_step` and `project_lstep` respectively) and finally *trace equivalence* for global trees and local environments (`TraceEquivalence`).

In `Proc.v` the *syntax and semantics of Zooid processes* are defined, together with the *typing system*, as a relation between processes and local types. The file contains the two central theorems on which Zooid's DSL is built: *type preservation* (`preservation`) and *protocol compliance*, in terms of obedience of execution traces of processes obeying the discipline of a global type (`process_traces_are_global_types`). Also, in the file `Proc.v`, the infrastructure for code extraction is given.

In `Zooid.v` we set up our DSL: *Zooid terms* are dependent pairs of a process and a proof that it is well-typed with respect to a given local type. The library provides smart constructors, helper functions and notations, in order to enable building these well-typed-by-construction terms.

Finally, in `Examples.v`, we display, through examples, the applicability of Zooid. We implement recursive pipeline, recursive ping-pong, simple calculator service, and the two buyer-protocol, a common benchmark of multiparty session types.

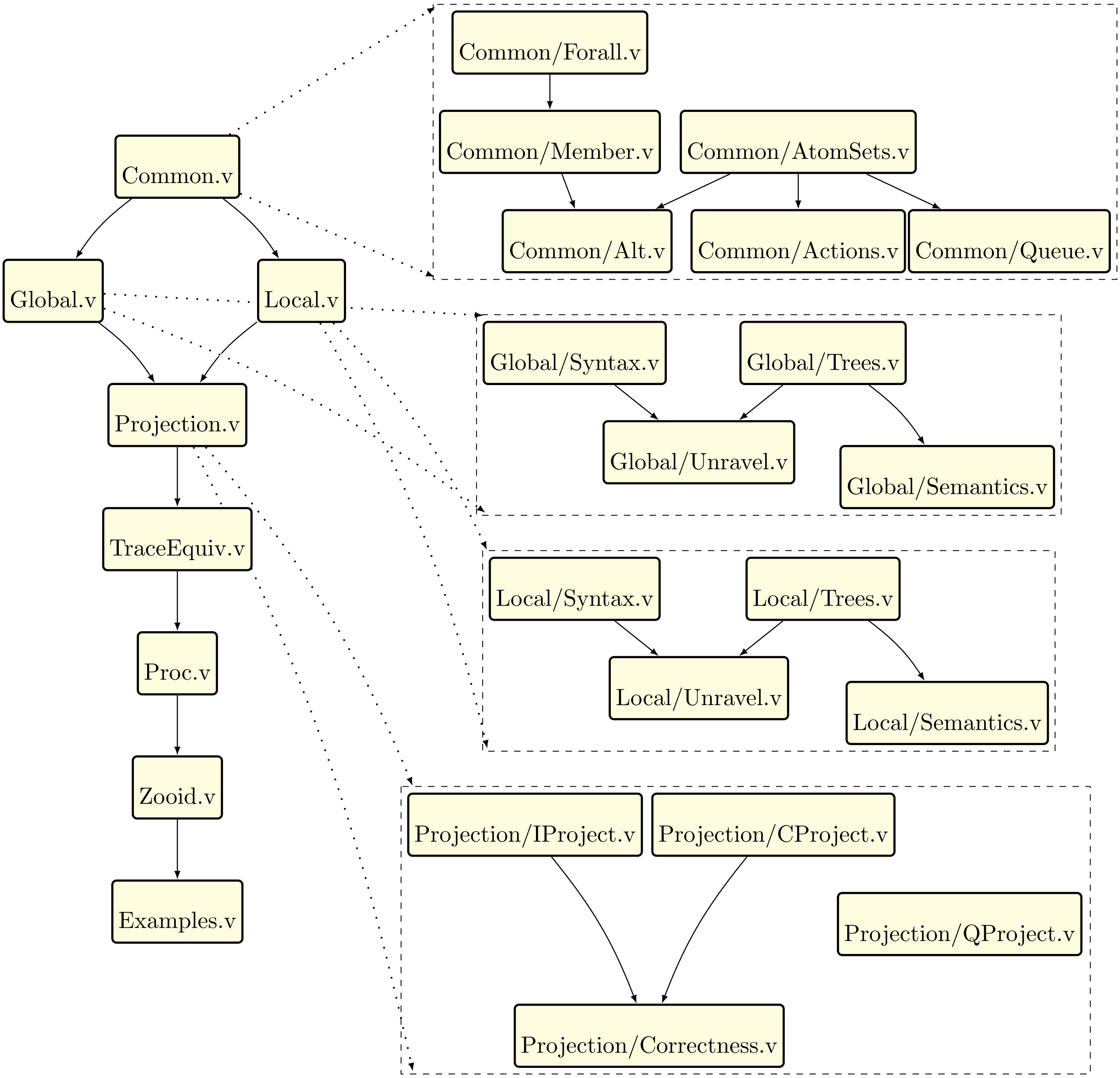

Figure 1: Dependency Graph for Zooid's Coq Libraries