Connectivity Graphs Artifact

ANONYMOUS AUTHOR(S)

This archive contains the artifact for "Connectivity Graphs: A Method for Proving Deadlock Freedom Based on Separation Logic". The contents of the archive are:

- The file "`cgraphs_sources.zip`" containing the source code.
- The file "`cgraphs_vm.ova`" containing the virtual machine.
- The file "`readme.pdf`", which is this document.

The source code is also available on GitHub at https://github.com/julesjacobs/cgraphs.

1 SETUP AND SANITY TESTING

This artifact contains our Coq development mechanizing the proofs in the paper. You can either use the Coq source code or you can use the VM in which Coq and the dependencies have already been installed.

1.1 Using the source code

The Coq development has been built and tested with the following dependencies

- A recent version of Coq (we tested with Coq 8.13.2)
- A development version of Iris (we tested with dev.2021-09-27.1.d2c226e7)
- A development version of std++ (we tested with dev.2021-09-27.0.7d5f3593)

You can compile the project as follows:

(1) Open a terminal in the `cgraphs_sources` folder
(2) Run the command `make`

You can also open the `.v` proof scripts in your Coq editor with support for stepping through the proof scripts, such as CoqIDE, Emacs with Proof General, or Visual Studio code with VSCoq.

The file `sessiontypes/safety.v` contains the final theorem, as well as instructions for having Coq check that the proof is complete using the `Print Assumptions` command.

1.2 Using the VM

To use our VM, please install VirtualBox. The VM has been tested with VirtualBox 6.1.26, which is the most recent version as of the time of this writing. The VM contains Ubuntu, Coq, CoqIDE, opam, Iris, std++, and the source code which has also been compiled.

On the desktop you will find a folder "cgraphs_sources". You can open the `.v` files in CoqIDE by running `coqide the_file.v` from the terminal. CoqIDE will allow you to step through the proof scripts.

The file `sessiontypes/safety.v` contains the final theorem, as well as instructions for having Coq check that the proof is complete using the `Print Assumptions` command.

The project has already been compiled for you, but you can let Coq re-check the entire development by performing the following steps:

(1) Open a terminal in the `cgraphs_sources` folder
(2) Run the command `make clean`
(3) Run the command `make`

2 EVALUATION INSTRUCTIONS

For evaluating the artifact, there are two main claims of the paper:

(1) **The Coq development mechanizes the results in the paper.** The final theorem is Theorem 3.2 from the paper, which is proved in `safety.v`. This file contains instructions on how to verify that the Coq proof is complete using the `Print Assumptions` command. In order to convince yourself that the `safety` theorem matches Theorem 3.2, please check that the language definition in Section 2 and the extensions in Section 5 match the definitions in Coq. The definitions and lemmas from the other sections are implementation details, because they only impact the *proof* of Theorem 3.2 (which Coq checks for you), and not the *statement* of Theorem 3.2. You may nevertheless want to verify that they match the lemmas in the paper. We have included a dictionary to translate between the paper and the Coq development in Section 4.
(2) **The proof has a layered structure**: the definitions in the `cgraph` directory are generic over the language definition. This is easy to verify, as none of the files in the `cgraph` directory import any of the files in the `sessiontypes` directory.

3 DIRECTORY STRUCTURE

The project consists of two parts: the generic connectivity graph library in `cgraphs/`, and the deadlock freedom proof for a session-typed functional language in `sessiontypes/`.

3.1 Connectivity graph library: `cgraphs/`

- Utilities and data structures:
  - `cgraphs/util.v`: miscellaneous utility functions and lemmas
  - `cgraphs/multiset.v`: multisets represented as lists up to permutations
  - `cgraphs/map_to_multiset.v`: conversion from maps to multisets of key-value pairs
  - `cgraphs/mapexcl.v`: utility function required for integration with Iris
- Separation logic:
  - `cgraphs/upred.v`: a linear version of Iris' uPred (Iris is affine)
  - `cgraphs/bi.v`: the bi interface to enable use of the Iris proof mode
  - `cgraphs/seplogic.v`: the instantiation of the separation logic and adequacy lemmas
- Undirected forests:
  - `cgraphs/uforests.v`: undirected, unlabeled forest library
- Connectivity graphs:
  - `cgraphs/cgraph.v`: the directed, labeled connectivity graph library and graph transformations
- Generic invariant and transformation lemmas:
  - `cgraphs/genericinv.v`: the generic invariant definition and separation logic graph transformations

3.2 Deadlock freedom proof: `sessiontypes/`

- Language definition:
  - `sessiontypes/langdef.v`: the definition of the session typed language, and sanity lemmas about the notions defined (e.g. the dual function and type equivalence)
- Run-time type system:
  - `sessiontypes/rtypesystem.v`: definition of the run-time type system, and lemmas related to it, such as substitution and decomposition into expression & context
  - `sessiontypes/langlemmas.v`: substitution in empty environment and type preservation under pure steps
- Invariant and preservation proof:
  - `sessiontypes/invariant.v`: definition and preservation of the invariant

- Global progress proof:
  - `sessiontypes/progress.v`: proof that the invariant implies progress
- Final theorem:
  - `sessiontypes/safety.v`: proof that preservation & progress together imply safety
- Y-combinator:
  - `sessiontypes/ycombinator.v`: definition and lemmas about the y-combinator

4 CORRESPONDENCE BETWEEN THE COQ DEVELOPMENT AND THE PAPER

We provide a dictionary to translate between the paper and the Coq development.

4.1 Definitions

Paper notation	Section	Coq notation	Coq location
Expr	2	`expr`	`sessiontypes/langdef.v`
Val	2	`val`	`sessiontypes/langdef.v`
Chan	2	`endpoint`	`sessiontypes/langdef.v`
Heap	2	`heap`	`sessiontypes/langdef.v`
Cfg	2	`list expr * heap`	`sessiontypes/langdef.v`
$\rightsquigarrow_{\text{pure}}$	2	`pure_step`	`sessiontypes/langdef.v`
$\rightsquigarrow_{\text{head}}$	2	`head_step`	`sessiontypes/langdef.v`
$\rightsquigarrow_{\text{global}}$	2	`step`	`sessiontypes/langdef.v`
Ctx	2	`ctx`	`sessiontypes/langdef.v`
Type	2	`type`	`sessiontypes/langdef.v`
Session	2	`chan_type`	`sessiontypes/langdef.v`
$\Gamma \vdash e : \tau$ (typing judgement)	2	`typed Γ e t`	`sessiontypes/langdef.v`
$Cgraph(V, L)$	3	`cgraph V L`	`cgraphs/cgraph.v`
V	3	`object`	`sessiontypes/rtypesystem.v`
L	3	`clabel`	`sessiontypes/rtypesystem.v`
Emp, $\ulcorner\phi\urcorner, \square P, \cdots$	3&5	`emp`, $\ulcorner\ulcorner\phi\urcorner\urcorner, \square P, \cdots$	`cgraphs/upred|bi|seplogic.v`
$\Gamma \vdash e : \tau$ (runtime judgement)	3	`rtyped Γ e t`	`sessiontypes/rtypesystem.v`
$\text{wf}(P)$	3	`inv f`	`cgraphs/genericinv.v`
$\text{wf}^{local}_{(\vec{e},h)}(v, \Delta)$	3	`state_inv`	`sessiontypes/invariant.v`
$\vdash_{buf} \vec{v} : (s_1, s_2)$	3	`buf_typed`	`sessiontypes/invariant.v`
$\text{active}(\vec{e}, h)$	3	`active`	`sessiontypes/progress.v`
$\text{blocked}_{(\vec{e},h)}(v_1, v_2)$	3	`waiting`	`sessiontypes/progress.v`
$(\vec{e}, h)$ can step	3	`reachable`	`sessiontypes/progress.v`
Undirected acyclicity	4	`cgraph_wf`	`cgraphs/cgraph.v`
unrestricted	5	`unrestricted`	`sessiontypes/langdef.v`
$\Gamma_1 \perp \Gamma_2$	5	`disj Γ1 Γ2`	`sessiontypes/langdef.v`
$\tau_1 \equiv \tau_2$	5	`type_equiv`	`sessiontypes/langdef.v`
Y	5	`y`	`sessiontypes/ycombinator.v`

4.2 Lemmas and theorems

Paper name	Section	Coq name	Coq location
Theorem 3.1	3	not applicable*	not applicable*
Theorem 3.2	3	`safety`	`sessiontypes/safety.v`
Lemma 4.1	4	`insert_edge_wf`	`cgraphs/cgraph.v`
Lemma 4.2	4	`delete_edge_wf`	`cgraphs/cgraph.v`
Lemma 4.3	4	`exchange`	`cgraphs/cgraph.v`
Lemma 4.4	4	`no_self_edge`	`cgraphs/cgraph.v`
Lemma 4.5	4	`edge_out_disjoint`	`cgraphs/cgraph.v`
Lemma 4.6	4	`cgraph_ind”`	`cgraphs/cgraph.v`
Lemma 5.1	5	`inv_exchange`	`cgraphs/genericinv.v`
Lemma 5.2	5	`inv_dealloc`	`cgraphs/genericinv.v`
Lemma 5.3	5	`inv_alloc_l`	`cgraphs/genericinv.v`
Lemma 5.4	5	`inv_alloc_r`	`cgraphs/genericinv.v`
Lemma 5.5	5	`inv_alloc_lr`	`cgraphs/genericinv.v`

∗ Theorem 3.1 is the type safety theorem for pure languages, and is stated only for illustration. Our language is not pure. The appropriate notion of type safety in our setting is Theorem 3.2.

4.3 Differences between the paper and the Coq development

There are only minor differences:

- The paper defines *Cfg* as a pair of a list of expressions and a heap. In Coq we do not use a pair, but we use curried definitions that take two arguments.
- Some graph lemmas are stated slightly differently in the paper, e.g., the `exchange` lemma. In the paper we say "let G, H ..." whereas in Coq the *lemma* provides H (there called `g'`) via the $\exists g'$. Since the conditions of the theorem uniquely determine H from G, they are equivalent. The Coq version is more convenient to use, because the lemma constructs `g'` for us, rather than the user of the lemma having to do that themselves.
- Since Coq's built-in equality does not work for coinductive types, we have to use setoids (the `Proper` type classes), and we have to prove that everything respects (coinductive) equivalence.
- The reviewers have asked us to prove a stronger progress statement, which we have done (`strong_progress` in Coq). Due to this, the line counts reported in Section 7 do not match the Coq development exactly any more. We will of course update the line counts in the final version of the paper, and we will also incorporate the stronger theorem statement in the paper. The Coq development still proves the older, weaker statement (`global_progress`) in terms of the stronger one.