Regrading Policies for Flexible Information Flow Control in Session-Typed Concurrency (Artifact)

Farzaneh Derakhshan
Illinois Institute of Technology, Chicago, IL, USA

Stephanie Balzer
Carnegie Mellon University, Pittsburgh, PA, USA

Yue Yao
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

This artifact is a Docker image containing the snapshot of the source code, a built command-line binary, and an interactive demonstration of the type-checker developed for IFC language of the main paper. This article discusses its scope, contents and methods of use.

2012 ACM Subject Classification Theory of computation → Linear logic; Security and privacy → Logic and verification; Theory of computation → Process calculi

Keywords and phrases Regrading policies, session types, progress-sensitive noninterference

Digital Object Identifier 10.4230/DARTS.10.2.4

Funding *Stephanie Balzer*: Supported in part by the Air Force Office of Scientific Research under award number FA9550-21-1-0385 (Tristan Nguyen, program manager). Any opinions, findings and conclusions or recommendations expressed here are those of the author(s) and do not necessari ly reflect the views of the U.S. Department of Defense.

Related Article Farzaneh Derakhshan, Stephanie Balzer, and Yue Yao, "Regrading Policies for Flexible Information Flow Control in Session-Typed Concurrency", in 38th European Conference on Object-Oriented Programming (ECOOP 2024), LIPIcs, Vol. 313, pp. 11:1–11:29, 2024.
`https://doi.org/10.4230/LIPIcs.ECOOP.2024.11`

Related Conference 38th European Conference on Object-Oriented Programming (ECOOP 2024), September 16–20, 2024, Vienna, Austria

Evaluation Policy The artifact has been evaluated as described in the ECOOP 2024 Call for Artifacts and the ACM Artifact Review and Badging Policy.

1 Scope

This artifact is a Docker [1] image containing the source code, built command-line binary and an interactive demonstration of a type-checker developed for the IFC language of the main paper. Additionally, it is pre-loaded with examples show-casing the syntax and features of the langauge.

The examples it provides include the *analyzer and surveyor* exmaple used in the main paper (Fig. 2, the type-checker accepts it) and its faulty variants (Fig. 3; type-checker rejects both based on failed synchronization pattern checks). It also includes the *banking* example in Fig. 9.

We list the main language features advertised and supported by the type-checker as follows:

- *User-specified security semilattice.* Example in `typecheck.ses`. The initial `secrecy ... end` block allow the user to define a secrecy lattice.
- *Lattice theories for polymorphic process definitions.* Example in `typecheck.ses`. Blocks `theory ... end` define theories `th1` to `th3` with varying amount of clauses and lattice variables. Later definitions of processes in `proc signature ... end` block use these theories as arguments.

- *Mutual recursive definitions.* Example in `typecheck.ses`. Block `session signature ... end` illustrates (mutual) recursive definitions of types and block `proc signature ... end` illustrates (mutual) recurive process definitions.
- *Synchronization pattern check.* This is exercised as part of the normal type checking process. Examples `analyze*.ses` provide code that passes or fails the checks. Source code `sillsec/statics.ml:118` implements the feature.

Follow instruction below to exercise the interactive demonstration:

- Load the image into a Docker container. Please consult the lastest Docker documentation.
- Once the image has been loaded, execute `docker run -p:8000:8000 ecoop24`. Make sure that host port `8000` is available to bind. The terminal outputs are for diagnostic purposes in case you encounter issues in the next step.
- In any modern browser, access `http://localhost:8000/`. An example has been pre-loaded for you. Click `Typecheck` on the bottom-left panel to run the type-checker. Type-checking should finish without error almost instantaneously.
- More examples are available from the drop-down on top left. After selecting an example, click `Load Example` to load it. The Left panel can be edited in place.

The interactive type-checker is built with `js_of_ocaml` [3]. It runs completely and locally in the browser.

To inspect the content of the image, to access the command-line binary, or to rebuild the project, fisrt obtain a shell in a running container. Execute `docker exec -it $CONTAINER_ID bash` where `$CONTAINER_ID` is the container ID of the container running the image. Consult lastest Docker document for comamnds obtaining this ID.

Once inside the container, further execute `eval $(opam env)` to load the `OCaml` development environment variables. From there you can:

- Run `make publish` to rebuild the project for both toplevels. Alternatively, the project can be built with the installed build manager `dune` [2].
- Run `dune exec top/top.exe < tests/bank.ses` to exercise the type-checker as a standalone command-line binary. The program takes its input from `stdin`.

2 Content

The artifact package includes a single docker image. The image contains the source code and its built output (as a executable and as a webapp) for the type checker of the paper.

Inside the image, all source code can be found under `/sintegrity`. We list the directory structure under `/sintegrity`:

- Sub-directory `sillsec` contains the type checker itself, including parser and lexer.
- Sub-directory `top` and `topjs` are binary / JS toplevels repsectively.
- Sub-directory `tests` contains test cases we have hand written. `tests/syntax.ses` and `tests/typecheck.ses` illustrates most of the syntax.

Here is a listing of the name of the test files under `tests` sub-directory:

- File `syntax.ses` illustrates the syntax of the langauge.
- File `typechecks.ses` show cases various features of the type-checker not demonstrated in separate examples.
- File `analyzer-surveyer.ses` contains the example in Fig. 2 of the main paper.
- Files `a-s-reckless.ses` and `a-s-hasty.ses` contains two incorrectly implemented versions of the files. The type checker throws an exception signifying a failed synchronization pattern check.
- File `bank.ses` contains example used in Fig. 9 in the main paper.

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). In addition, the artifact is also available on Zenodo at: `https://zenodo.org/records/12735388`.

4 Tested platforms

This artifact is developped and tested on an `x86-64` Linux machine, running WSL2-Ubuntu. The artifat evaluation has also been carried out on an Apple M2 (`aarch64`) platform. Any reasonably mordern hardware, capable of running OCaml 4.14 compiler, on a Linux x64 platform should be sufficient to run and continue developping this artifcat.

This artifact contains an interactive demonstration. Any resonably mordern browser should be sufficient for the demonstration. Once the Docker image has been imported, the demonstration should be ready within a minute.

5 License

6 MD5 sum of the artifact

e08e665e2106cb240317fb6524b2d398

7 Size of the artifact

1.36 GiB

References

1 Docker: Accelerated container application development. `https://www.docker.com/`. Accessed: 2024-07-12.

2 Dune: A composable build system for ocaml. `https://dune.build/`. Accessed: 2024-07-12.

3 `ocsigen/js_of_ocaml`: Compiler from ocaml to javascript. `https://github.com/ocsigen/js_of_ocaml`. Accessed: 2024-07-12.