Linear Promises: Towards Safer Concurrent Programming (Artifact)

Ohad Rau
Georgia Institute of Technology, Atlanta, GA, USA

Caleb Voss
Georgia Institute of Technology, Atlanta, GA, USA

Vivek Sarkar
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

We present a compiler for a concurrent programming language, which utilizes linear typing to create a safer promise abstraction. The compiler is implemented in OCaml and produces source-level Java code. We provide example programs to demonstrate use of the language, as well as translations of incorrect JavaScript code from StackOverflow to showcase the ability of our language to catch many classes of bugs. Finally, we provide a minimal runtime environment to allow the execution of compiled programs.

2012 ACM Subject Classification Software and its engineering → Concurrent programming languages; Theory of computation → Operational semantics; Theory of computation → Type theory

Keywords and phrases promises, type systems, linear typing, operational semantics, concurrency

Digital Object Identifier 10.4230/DARTS.7.2.15

Related Article Ohad Rau, Caleb Voss, and Vivek Sarkar, "Linear Promises: Towards Safer Concurrent Programming", in 35th European Conference on Object-Oriented Programming (ECOOP 2021), LIPIcs, Vol. 194, pp. 13:1–13:27, 2021. `https://doi.org/10.4230/LIPIcs.ECOOP.2021.13`

Related Conference 35th European Conference on Object-Oriented Programming (ECOOP 2021), July 12–16, 2021, Aarhus, Denmark (Virtual Conference)

1 Scope

The artifact implements the compiler described in the paper (described in Section 4). First and foremost, the compiler can be used to demonstrate the language's type system and semantics. This allows for general programming using the system and can demonstrate the programming model. The included examples can be used to reproduce the type checker benchmarks (from Section 5.1). Additional examples, taken from StackOverflow, can be used to reproduce the bug-catching case study (from Section 5.2).

2 Content

The artifact package is a Docker image (saved as a gzipped tar file). The compiler is pre-installed as `linear-promises.compiler`, along with all of its dependencies and OpenJDK 11. In addition, the image includes the following files:

- `README.md`: Overview document explaining the basic usage and command-line interface for the compiler. Includes instructions for running the examples, accessing the case study's StackOverflow excerpts, and benchmarking the example programs. In addition, an overview of the language's syntax is included.

- `examples/`: Set of example programs that were used to test the compiler. These programs showcase different ways of using the language, and can be used to replicate the compiler benchmarks from Section 5.1.
- `stackoverflow/`: Set of programs translated from StackOverflow questions in the case study (Section 5.2). Each program's name corresponds to a certain StackOverflow question ID, and the results in Table 2 indicate whether the type checker is expected to reject each program.
- `src/`: Contains the OCaml source code for the compiler, consisting of:
  - `compiler.ml`: Parses command-line arguments and runs the compilation pipeline.
  - `contexts.ml`: Implements the data structures used to provide typing contexts (specifically, typing environments & sets of variables).
  - `ident.ml`: Utilities to allow variable shadowing in generated Java code.
  - `javaCG.ml`: Implements Java code-generation via translation from our language's AST to source-level Java and provides built-in methods (e.g. file I/O) used in the examples.
  - `lang.ml`: Defines the syntax tree for the language.
  - `lexer.mll`: Tokenizer for the language, utilizing the OCamllex lexer generator.
  - `parser.mly`: Parser for the language, utilizing the Menhir parser generator.
  - `typecheck.ml`: The type-checking code for the language. This is the bulk of the compiler and is described in pseudo-code in the paper (section 4.2).
- `runtime/src/`: Contains the Java source code for the runtime library and serves as the environment for running the example programs. This consists of:
  - `numbers.txt`, `hello.txt`: Input to programs demonstrating file I/O.
  - `lang/promises/`: Directory containing the runtime's source code.

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 at: `https://github.com/OhadRau/LinearPromises/releases/download/v1.0/artifact.tar.gz`.

4 Tested platforms

The artifact has been tested on Windows 10 (using WSL2 running Debian 10) and Alpine Linux 3.13. The system depends on OCaml and Java, and has been tested against OCaml 4.11.1 and 4.12.0, as well as OpenJDK 11.0.5 and 11.0.9. The system has been tested with as little as 1 CPU core and 1GB of memory.

For cross-platform compatibility, the package is distributed as a Docker image (running Alpine 3.13, OCaml 4.12.0, and OpenJDK 11.0.9). This image should be compatible with all major platforms, provided the system fulfills the minimum requirements of Docker, OCaml and OpenJDK.

Starting the Docker Image

To run the Docker image, you must first fetch the artifact archive. Then, run the following commands to load it into Docker and start the image:

```
$ docker load ./artifact.tar.gz
$ docker run -it ohadrau/linear-promises:latest
```

5 License

The artifact is available under the MIT license (`https://opensource.org/licenses/MIT`).

6 MD5 sum of the artifact

f6ecd59c832dbe016d2bf78256bc96d9

7 Size of the artifact

743 MiB