Sound and Reusable Components for Abstract Interpretation

Artifact

SVEN KEIDEL and SEBASTIAN ERDWEG, JGU Mainz, Germany

1 ARTIFACT DESCRIPTION

The artifact contains the library of analysis components (Section 6) and the code of the case studies (Section 7). More specifically, the results of the paper can be found in the following folders and files in the artifact:

Section 6 The library of analysis components can be found in the folder lib/. The interfaces of a components can be found in lib/src/Control/Arrow/ and the concrete and abstract instance of these components in lib/src/Control/Arrow/Transformer/{Concrete,Abstract}.

Section 7.1 The generic interpreter of the While language can be found in while/src/GenericInterpreter.hs.

Section 7.1.1 The interval analysis of the While language can be found in while/src/IntervalAnalysis.hs.

Section 7.1.2 The reaching definition analysis of the While language can be found in while/src/ReachingDefinitionsAnalysis.hs,

Section 7.1.3 The analysis of the While language extended with exceptions can be found in while/src/Exceptions/ReachingDefinitionsAnalysis.hs.

Section 7.2 The generic interpreter and the k-CFA analysis of PCF can be found in the files pcf/src/GenericInterpreter.hs and pcf/src/IntervalAnalysis.hs.

Section 7, Tables 1-4 The tables give an overview of the analysis components that the case studies use. The interfaces of a components can be found in lib/src/Control/Arrow/ and the concrete and abstract instance and liftings of these components in lib/src/Control/Arrow/Transformer/{Concrete,Abstract}.

2 GETTING STARTED GUIDE

The artifact is open-source and can be obtained by cloning the following git repository

```
git clone https://github.com/svenkeidel/sturdy --branch oopsla-19-artifact
cd sturdy
```

The generated HTML documentation for the artifact, can be found in the docs folder.

The source code of the artifact is written in Haskell and can be compiled and tested with the *Stack* build tool (https://www.haskellstack.org/). To setup the Haskell compiler run `stack setup` and to compile and test the code of the artifact run the following commands:

```
stack build --fast   # --fast turns off compiler optimizations which speeds up builds.
stack test --fast
```

WARNING: Executing these commands for the first time can take a while, because they need to download and compile all dependencies of the artifact.

Authors' address: Sven Keidel; Sebastian Erdweg, JGU Mainz, Germany.

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

Additionally, we packaged the artifact in a docker container. You can find instructions on how to install docker at https://docs.docker.com/get-started/. To download and start the container run the following commands:

```
docker pull svenkeidel/oopsla-19-artifact
docker run -it svenkeidel/oopsla-19-artifact
```

WARNING: The docker image is 10GB large and downloads over slow internet connections might take a while. Hence, we recommend to browse the code of the artifact in the git repository.

3 STEP BY STEP INSTRUCTIONS

We suggest that the artifact reviewers inspect the source code of the analyses and compare it to the results in the Tables 1-4.

- A box $\boxed{\text{n}}$ in the table means that the analysis component $\langle Row_L, Row_R \rangle_{Col}$ implements the interface *Col* with *n* operations. For example, in Table 1 the box $\boxed{4}$ in the cell $\langle \texttt{EnvT}, \widehat{\texttt{EnvT}} \rangle_{\texttt{ArrowEnv}}$ means that that the concrete arrow transformer `EnvT`[1] and abstract arrow transformer $\widehat{\texttt{EnvT}}$[2] implement the interface `ArrowEnv`.[3]
- A straight arrow ↑ represents a trivial lifting (Section 4.2) through a component. For example, in Table 1 the straight arrow at the cell $\langle \texttt{EnvT}, \widehat{\texttt{EnvT}} \rangle_{\texttt{ArrowStore}}$ means that the `EnvT` and $\widehat{\texttt{EnvT}}$ transformers lift the `ArrowStore` operations. Most of the trivial liftings can be automatically derived by Haskell:

  ```
  newtype EnvT = ... deriving (ArrowStore, ...)
  ```

- A squiggly arrows ↭ represents a non-trivial lifting (Section 4.2). For example, in Table 1 the squiggly arrow at the cell $\langle \texttt{FailureT}, \widehat{\texttt{FailureT}} \rangle_{\texttt{Arrow/ArrowChoice}}$ means that the arrow transformer `FailureT` and $\widehat{\texttt{FailureT}}$ lift the `Arrow` and `ArrowChoice` operations. The lifting is non-trivial, because the arrow transformer use the `KleisliT` transformer, that use the `mapA` and `mapJoinA` functions for the `Error`[4] and the $\widehat{\texttt{Error}}$[5] type. In fact, all arrow transformers that are implemented with the `KleisliT` transformer have non-trivial liftings that require an explicit soundness proof.

[1] lib/src/Control/Arrow/Environment.hs
[2] lib/src/Control/Arrow/Transformer/Concrete/Environment.hs
[3] lib/src/Control/Arrow/Transformer/Abstract/Environment.hs
[4] lib/src/Data/Concrete/Error.hs
[5] lib/src/Data/Abstract/Error.hs