Typed First-Class Traits (Artifact)

Xuan Bi[1]
The University of Hong Kong, Hong Kong, China
xbi@cs.hku.hk

Bruno C. d. S. Oliveira[1]
The University of Hong Kong, Hong Kong, China
bruno@cs.hku.hk

Abstract

This artifact contains the prototype Haskell implementation of SEDEL, with support for first-class traits, as described in the companion paper. This artifact also contains the source code of the case study on "Anatomy of Programming Languages", illustrating how effective SEDEL is in terms of modularizing programming language features. For comparison, it also includes a vanilla Haskell implementation of the case study without any code reuse.

2012 ACM Subject Classification Software and its engineering → Object oriented languages
Keywords and phrases traits; extensible designs
Digital Object Identifier 10.4230/DARTS.4.3.1
Related Conference 32nd European Conference on Object-Oriented Programming (ECOOP 2018), July 19–21, 2018, Amsterdam, Netherlands

1 How to Use

The artifact contains the reference interpreter of SEDEL in Haskell, as described in the paper "Typed First-Class Traits". We document in detail how to build and run the interpreter, as well as the examples and case study from the paper.

1.1 Building Instructions

This project can be built with *cabal* or *stack*. We strongly recommend using the stack build tool. Though cabal may work (we haven't tested), the following instructions assume you use the stack build tool.

1.1.1 Prerequisites

Install *the Haskell toolchain*. The easiest way is via this page: `https://www.haskell.org/downloads`. We recommend the "minimal installers". Choose a suitable installer according to your platform. After the installation, make sure `stack` can be invoked (type `stack` in the terminal, if you see "command not found" this means you have not properly installed `stack`).

1.1.2 Build and Run

1. Unzip the archive file
2. Go to the `impl` directory
3. Type `stack setup` in the terminal, which will download the GHC compiler of the version that was used in our development. This may take a bit of time depending on your Internet connection.

[1] Funded by Hong Kong Research Grant Council projects number 17210617 and 17258816

4. Type `stack build` to build the project, you should see something like the following (suppose `~` is the prompt):

```
~ stack build
...
Building all executables for ‘sedel’ once. After a successful build of all of them...
sedel-0.1.0.0: configure (lib + exe)
Configuring sedel-0.1.0.0...
sedel-0.1.0.0: build (lib + exe)
Preprocessing library for sedel-0.1.0.0..
Building library for sedel-0.1.0.0..
[ 1 of 13] Compiling Paths_sedel            (...)
[ 2 of 13] Compiling SEDEL.Common           (...)
[ 3 of 13] Compiling SEDEL.Source.Syntax    (...)
[ 4 of 13] Compiling SEDEL.PrettyPrint      (...)
[ 5 of 13] Compiling SEDEL.Environment      (...)
[ 6 of 13] Compiling SEDEL.Target.Syntax    (...)
[ 7 of 13] Compiling SEDEL.Target.Eval      (...)
[ 8 of 13] Compiling SEDEL.Util             (...)
[ 9 of 13] Compiling SEDEL.Source.Desugar   (...)
[10 of 13] Compiling SEDEL.Source.Subtyping (...)
[11 of 13] Compiling SEDEL.Source.TypeCheck (...)
[12 of 13] Compiling SEDEL.Parser.Parser    (...)
[13 of 13] Compiling SEDEL                  (...)
Preprocessing executable 'sedel-exe' for sedel-0.1.0.0..
Building executable 'sedel-exe' for sedel-0.1.0.0..
[1 of 2] Compiling Main                     (...)
[2 of 2] Compiling Paths_sedel              (...)
Linking .stack-work/dist/x86_64-osx/Cabal-2.0.1.0/build/sedel-exe/sedel-exe ...
sedel-0.1.0.0: copy/register
Installing library in ...
Installing executable sedel-exe in ...
Registering library for sedel-0.1.0.0..
```

5. Type `stack exec sedel-exe`, you should see the following:

```
~ stack exec sedel-exe
SEDEL, version 0.1, :? for help
>
```

6. Now you are in the REPL of SEDEL, where you can enter expressions to type check and evaluate. The REPL prompt is `>`, and
   - `:q` to quit
   - `:load` to load a file
   - `:?` for usage

   Below is some example use:

```
> :load examples/case_study.sl
Typing result
: String

Evaluation result
=> "add1(var x = 3.0; var y = 4.0; (if (x < y) then (x + 2.0) else (y + 3.0))) = 6.0"
```

1.2 Language Reference

A program consists of list of declarations (separated by `;`), ended with a `main` declaration. Like Haskell, a line comment starts with `-` and a comment block is wrapped by `{-` and `-}`.

Note that due to some parsing problem, if you want to try out some expression directly in the REPL, you need to add `main =` before the expression. For example, `main = true ,, 3` will work, whereas `true ,, 3` alone will be a parsing error. It is recommended to write SEDEL programs in a file, and then load it in the REPL to test.

- Primitive type: `Int`, `Bool`, `Double` and `String`
- Top type/value: `() : Top`
- Type annotation: `2 : Int`
- Merge: `true ,, 3`
- Intersection type: `Bool & (Int -> Int)`
- If expression: `if x == 0 then true else false`
- Term abstraction: `(\x -> x+1) : Int -> Int`
- Type abstraction: `/\ (A * Int) . (\x -> x) : A -> A`
- Disjoint (universal) quantification: `forall (A*Int) . A -> A`
- Term declaration: `id A (x : A) = x`
- Type declaration: `type Person = {name : String, male : Bool}`
- Traits: `trait [self : Person] => { age = 42 }`

Refer to `*.sl` files in the `examples` directory to learn more about the SEDEL syntax.

1.3 Examples and Case Study

Table 1 shows how each of the examples in the paper corresponds to the SEDEL files in the artifact. You can use the `:load` command in the REPL to see the results.

Table 1 Paper-to-artifact correspondence

Examples	Paper	File
SEDEL examples, mixin style	Page 7, Section 2.2	`examples/overview2.sl`
SEDEL examples, trait style	Page 10, Section 3	`examples/overview2.sl`
Object Algebras in SEDEL	Page 20, Section 5.1	`examples/application.sl`
Case study	Page 22, Section 5.2	`examples/case_study.sl`