Design Principles of Programming Languages

Practices in Class

Chap 13-17

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong

Peking University, Spring Term, 2020

Code packages

- “fullref”
- “fullerror”
- “rcdsub”
- “fullsub”
- “joinsub”
- “joinexercise”

Practice #1

- Do exercise 17.3.1
  - The *joinexercise* typechecker is an incomplete implementation of the simply typed lambda-calculus with subtyping, records, and conditionals: basic parsing and printing functions are provided, but the clause for TmIf is missing from the typeof function, as is the join function on which it depends. Add booleans and conditionals (and joins and meets) to this implementation.
  - Refer to: § 16.3 showed how adding booleans and conditionals to a language with subtyping required extra support functions for calculating the least upper bounds of a given pair of types. The proof of Proposition 16.3.2 (see page 522) gave mathematical descriptions of the necessary algorithms

Practice #2

- Do exercise 17.3.3
  - the subtype check in the application rule fails, the error message that our typechecker prints *may not be very helpful* to the user. We *can improve it* by including the *expected parameter type* and *the actual argument type* in the error message.
  - Error reporting can be greatly improved by changing the subtype function so that, instead of returning true or false, it either returns a trivial value (the unit value ()) or *else raises an exception*.
  - Reimplement the typeof and subtype functions to make all of the error messages as informative as possible.

Design Principles of Programming Languages

Practices

Chap 18-19

Please refer to the package of “fullref”

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong

Peking University, Spring Term, 2020

Practice #1

- Do exercise 18.6.1
  - Write a subclass of resetCounterClass with an additional method dec that subtracts one from the current value stored in the counter
  - Use the fullref checker to test your new class

Practice #2

- Do exercise 18.7.1
  - Define a subclass of backupCounterClass with two new methods, *reset2* and *backup2*, controlling a second "backup register." This register should be completely separate from the one added by backupCounterClass: calling *reset* should restore the counter to its value at the time of the last call to *backup* (as it does now) and calling *reset2* should restore the counter to its value at the time of the last call to *backup2*.
  - Use the fullref checker to test your new class

Practice # 3 (Option)

- Do exercise 19.4.3
  - The operation of *assigning a new value to the field* of an object is omitted from FJ to simplify its presentation, but it can be added without changing the basic character of the calculus very much.
  - Using the treatment of references in Chapter 13 as a model.

Practice #4: Challenge (Option)

- Do exercise 19.5.5
  - Starting from one of the lambda-calculus typecheckers, build a typechecker and interpreter for Featherweight Java.
  - Submit your typechecker and interpreter before June 3