Part I Project:
Synthesis for Reliable and Interpretable AI
Fall 2017

In this project, you will implement a synthesizer from input-output examples. Best project will receive an award.

Project Details Implement a synthesizer that solves data-wrangling tasks from examples. The program space of the synthesizer is defined by a domain-specific language (DSL).

Data wrangling task: an example In data-wrangling tasks, outputs are concatenation of substrings of their inputs and constant strings. For example: The input-output example "Jane Doe $\rightarrow$ JD" defines the task of computing a person's initials, while the input-output example "Jane Doe $\rightarrow$ Hello J. Doe" defines the task of generating a customized greeting.

DSL We split the project into two parts: one that synthesizes programs over a simple DSL and one over a complete DSL, which subsumes the simple DSL.

Types Both DSLs support three types: strings, integers, and Boolean. Each type can be used to define variables, constants, and expressions.

Operators The simple DSL supports the following operators: string concatenation, extraction of substrings, equality check, addition, and less-than. In addition, it supports the ITE (if-then-else) operator, which consists of a condition c and two instructions i_1 and i_2, and its semantics is: if c is satisfied, execute i_1; otherwise, execute i_2. Programs over the simple DSL can be seen as compositions of operators. For example:

ITE(isEqual("a",substring(x,0,1)), concat("Hello", x), concat("Bye", x))

where x is the input string.

The complete DSL additionally supports assignments, sequential statements, loop statements, and return statements. You can extend the complete DSL with more operators, should you need to. Note, however, that: (1) you are not allowed to change the existing instructions and (2) your new instructions *should not* implement **SimpleDSL** (see code).

Implementation Skeleton We provide a skeleton of the DSLs and a few data-wrangling tasks (see attached zip). The skeleton consists of:

- The **instructions** package, containing the instructions of the DSL. Here, the sub package **advancedinstructions** contains the additional instructions of the complete DSL
- The **synthesizer** package, contains **Main.java**, **Synthesizer.java**, **SimpleDSLExamples.java** and **CompleteDSLExamples.java**.
  - **Main.java** contains an end-to-end example of a generation of a program over the DSL (see **exampleProg**) and a checking procedure (called **check**) which checks whether the synthesized program is consistent with the provided examples. The **main** function of **Main.java** consists of a call to execute the end-to-end example and two calls to the two parts of the competition: synthesis over the simple DSL and synthesis over the complete DSL.
  - **Synthesizer.java** is the class responsible for generating programs over the DSL, which is now empty and which you should implement. The **synthesize** function takes as parameters the input-output examples, the input variables, and the maximum timeout (in seconds) the synthesizer is allowed to run. Make sure you do not exceed the time limit.

Checking Your Synthesizer To check whether a synthesized program is correct on some examples, you can invoke the function **check** in **Main.java**, which will test your synthesized program on a set of examples. We have included some examples for you in **SimpleDSLExamples.java** and **CompleteDSLExamples.java** but these classes also contain empty methods for you to instantiate and provide more examples. We will also test your synthesizer on additional examples.

How to Start? Take a look at slides $7-12$ of Lecture 3, and think how characters in the output can (or cannot) be explained given the input. You are not obliged to use the technique shown in these slides. The language of use will be Java.

Competition The competition will consist of two parts, one for each DSL. In each part, we will run your synthesizer on data-wrangling examples (not all are included in the zip), and check how many examples are solved and how fast the synthesizer completes. You can choose to compete only in the first part of the competition.

Deliverables

1. A 3-page write-up PDF that explains at a high-level the approach you have taken for the synthesis process, including demonstration on few examples.

2. A zip containing the entire source package (including both packages), in particular it should contain your synthesizer and the missing examples in **SimpleDSLExamples** and in **CompleteDSLExamples** (if you choose to compete in the second part).

3. A script that compiles your code and executes **competition1** and **competition2** (if you choose to participate in the second part) and a README file explaining how to compile your code and run it (we will compile your code on a Linux machine).

Submission Instructions Submit the zip file to Dana Drachsler-Cohen at `dana.drachsler@inf.ethz.ch` (and also any questions you may have). Submissions that do not compile will not be eligible to participate in the competition and may not be reviewed. Submissions are allowed in groups of up to three students.

Submission deadline December 1st, 2017, AoE.