Constraint Synthesis & CEGIS

The purpose of this project is to extend the results of Exercise 3 with synthesis capabilities. There is more room for creativity and initiative in the final project relative to the homework exercises: you will need to design the suitable user interface (i.e., input and output formats), and you are welcome to devise new language features or algorithmic improvements. You will also create your own benchmark suite for demonstrating your tool's efficacy.

The project will be judged by the set of features, how well they are implemented, and mostly by your synthesizer's ability to generate interesting programs from non-trivial structural specifications.

Feature #1

The first step you will have to do is to extend the input language (while_lang) to a language of sketches by introducing an *integer hole* — with the semantics of `??` in Sketch (see the lecture slides and lab).

Then, introduce behavioral specifications in the form of PBE, and implement a constraint synthesis procedure that will find suitable assignments for holes in sketches such that the program will satisfy all input/output examples provided in the input. If no such assignment exists, the tool should reject the sketch with an appropriate error message.

After you have found the assignment, fill in the holes in the program and run the verifier from exercise 3.

Feature #2

Allow your tool to accept specifications in the form of `assert` statements, *without* the need to provide input-output examples. Like in Sketch, the synthesizer must fill in the holes such that the assertions are true for every input.

You will find that this is pretty hard to do when the assertions are inside or after a loop. Start by limiting this feature for scenarios where the program does not contain loops or where the `assert`s occur before the first loop.

Additional Features

Features #1 and #2 are nice but the setting is quite poor due to limitations of the programming language and of the assertion logic. Your task is to extend the functionality in one or more interesting way.

Here are some initial suggestions:

- More language features; emphasis on the ability to define data structures and use arrays. (Tip: Z3 has extensive support for arrays in first-order formulas. Consult the relevant documentation.)
- A techinque to handle loops in synthesis, to some extent. (Tip: you will probably have to bound the number of loop iterations, like Sketch does. By default, Sketch unrolls loops in sketches to 8 iterations.)
- Alternative methods for behavioral specification. (Use what we learned.)
- Interactivity features—give more feedback to the user and allow iterative development of specifications. This may include some kind of UI.
- More expressive synthesis—break the barrier of holes being only constants. Synthesize some arithmetic or array expressions. You may add as a secondary input a grammar for the synthesized pieces.

- Improve the performance with various heuristics and evaluate the improvement (compare performance with and without your optimization).

Documentation and Testing

Project submissions must include, in addition to the code, A README file with detailed instructions on how to run the tool and a description of the most interesting cases that you were able to synthesize successfully.

You must provide a test suite, as large and comprehensive as you can, demonstrating that the tool works. You may also include cases were the tool detects that the sketch is unrealizable, as well as cases where the tool is too slow or fails to synthesize (e.g. diverges) even though a solution exists.

The project will be graded according to the quality of the solved cases, the tool's performance, its ease of use, and the state of the documentation.

The testing suite and the documentation will account for 15% of the grade. The additional features you implement will account for an additional 15%. That is, a perfect implementation of features #1 and #2 with excellent testing and documentation will get you as far as 85. Invest in the extra features if you strive for 90 and above.