6.885 Project Proposal: Synthetic Software Evolution

James Koppel
Chelsea Voss

Description of the Problem

The design of data structures, file formats, wire protocols and other persistent parts of software design can have huge and long-lasting consequences on software quality. In particular, for many pieces of software, any design decision made in creating a file format can have permanent consequences, in that new versions of the software must be backwards-compatible and capable of reading the formats created by all previous versions of the software.

Depending on the design, backwards-compatibility may be impossible without extremely complicated and clunky code. We aim to prove this using synthesis.

Approach and Challenges

For now, we will focus on the design of file formats; in the future, we might consider how to apply this approach to database designs or API designs.

Consider each possible file format design as a node in a tree. Updating the current file format to a new design produces descendants of the current node – new file format designs. When we continue updating a system, we trace a path through this tree, eventually arriving at a final file format design. The code to interpret this final file format must also be compatible with all of the file format's ancestors in earlier designs; because of this, a series of poor design decisions might make the code unnecessarily complex, even if the final file format is simple.

To analyze the complexity of the code that would result from a series of past design decisions, we can use program synthesis to generate the program that would read each of the ancestor file formats.

One of the challenges of this project will be figuring out how to represent file formats and changes in file formats in a way that is abstract enough to be relevant to real-world systems, and yet still efficiently enumerable. It would be interesting to be able to take a real example of a system where file formats (or, most likely, database schemas) changed over time during development and analyze that system.

A possible extension to this project: also generate programs that can migrate information between file formats in the history of file formats.

Current Progress

Also included with this document are several source files containing our proof of concept.

one_field.sk contains our concept for the potential generated sketches. write1 and write2 model two file format writers as functions which output into a buffer, the first outputting one field, and the second two. This models the idea of adding a field to the file format. The goal is to sketch a read function that can read a buffer produced by either write function. To prevent it from filling in nonsense for fields not stored in a file format, our check function asserts that it return a default value (namely 0) for any fields not present in the original.

GenFormat.hs is our first implementation of a generator of trees of file formats. When run, it outputs a sequence of lines. Each line contains a list of file format specifiers modeling a history of file formats, each one created from the previous by adding or deleting a field. We plan on feeding the output of this program to another program which then generates a sketch for each file format history. Doing so allows us to easily parallelize the problem, using multiple servers to generate code for all format histories. (We foresee that the total number of histories may be large; potentially greater than 1 million. However, each sketch is independent, and we aim to keep individual solution times fast.)

State of the Art

We are not aware of any relevant work.

Significance

This project was conceived as a way to test Jimmy's thesis that certain ways of evolving file formats necessarily lead to complicated code because their greatest lower bound (types-theoretically) will be extremely complicated.

We also hope that tools of this form can effectively teach programmers the consequences of certain design decisions without them needing to have the experience of making the decisions poorly. We plan to release the output of this project as an interactive visualization and incorporate it into a blog post.

We also show a new application of synthesis: generating large quantities of simple programs to explore a design space. To our knowledge, this is the first use of synthesis where the challenge is to generate many programs instead of detailed programs.