LIVS: Language-Independence Via Synthesis

I propose, in order to fulfill my CPSC 490 Senior Project requirement, to complete the LIVS tool (Language Independence Via Synthesis), a program synthesis engine written in Haskell with the aim of being language-agnostic and accessible without sacrificing efficiency. I will complete this project advised by Ruzica Piskac and in collaboration with her and two of her graduate students, Bill Hallahan and Mark Santolucito.

The basic idea of program synthesis is to generate code (or fix bugs in existing code), based on a specification, so that the developer does not have to write the code or fix themselves. A specification, in one form or another, is a conjunction of promises about what the code will do. One of the simplest specifications to understand is a series of I/O examples, e.g. the synthesized function must output “Hello” when provided 3 as an input. It is immediately apparent that this kind of synthesis has the potential to be amazingly practical.

As synthesis techniques and engines become faster, more robust, and able to generate wider classes of programs, one of the major barriers to true, effortless practicality is the accessibility of synthesis. Existing state of the art synthesis engines rely on deep semantic knowledge of a particular programming language, and are restricted to use for that language. Moreover, these engines use techniques like symbolic execution and deeply nuanced static analysis, which, while powerful, might require strictly typed environments or rely on other rigid language features to work properly.

LIVS is a synthesis engine which pursues accessibility – synthesis should work for modern untyped languages, and be easily extensible to many programming languages. The Programming by Example (PBE) paradigm, where examples provide the entire specification for synthesis, should substitute formal specs for a human readable and writable set of guarantees. These features together would make LIVS ripe for practical application, and a novel and necessary contribution to the wealth of existing synthesis research.
This also presents unique challenges for LIVS – in particular, the engine needs to synthesize programs with no semantic information about the language in which the programs are written. LIVS learns the relevant subset of semantics necessary for a given synthesis problem by calling an interpreter on the available component functions with fuzzed input, collecting the output, and using a SyGuS solver to approximate formal definitions for those functions, which constitute a grammar for synthesis. Thus LIVS also only needs a minimal syntactic understanding of a language, enough to find functions in a code file and call them in an interpreter.

My proposal, as I stated, is to complete LIVS. The codebase already exists – I joined Ruzica’s group in working on it over the previous summer, before which Mark and Bill had already done substantial work on it. Nevertheless, there is still much to be done to make LIVS into an effective tool and moreover this work is:

1) Large enough in scope to constitute a semester-long project.
2) Theoretically deep enough to require that I expand my understanding of formal methods and solver-aided programming in order to complete it.

I am qualified to complete this work not only because I am already acquainted with the LIVS codebase, but because I received a baseline theoretical understanding off of which to build from Professor Piskac’s class, CPSC 554: Software Analysis and Verification.

I will now detail what has already been done for this project, what work is left to be done, and why this work fulfills 1) and 2) above. In its current state, LIVS:

- Is over 2000 lines of Haskell.
- Synthesizes a purely functional JavaScript definition over strings and integers via the above procedure, given a simple JS code file and examples on strings and integers.
- Uses some primitive methods to prune the grammar by types and reduce its size.
- Presents synthesis as a single large problem to CVC4, a SyGuS solver.
- Uses fuzzing to randomly generate examples for component functions.
- Lacks support for other programming languages.
- Lacks substantive benchmarks or performance analysis.
- Demonstrates inconsistent behavior for small examples and sometimes fails inexplicably.
- Does not scale to complex examples.
My work will address the deficiencies above and expand LIVS’s capability to the realm of the useful. In particular, my tasks, among any others that we discover along the way, are:

- Develop a varied set of benchmarks against which to test changes and improvements to LIVS, and to use for substantive analysis of LIVS’s performance.
- Explore methods of slicing up synthesis problems into subproblems to improve CVC4’s scalability and reliability.
- Develop new techniques to prune the synthesis grammar as aggressively as possible to speed up solving.
- Outline any and all specific guarantees that we can make about LIVS’s performance, and the class of programs which it will handle well, through analysis of the capabilities of SyGuS solvers.
- Further, design LIVS to pose problems to CVC4 in a regular and well-defined way so that we can make as many of these consistency guarantees as possible.
- Explore the effectiveness of example generation techniques besides fuzzing, which has been in practice not very useful.
- Expand language support to popular languages such as Python, and multiple others, to showcase the “language-independent” side of LIVS.
- Potentially find a way for LIVS to synthesize programs with loops and conditional expressions in them.
- Potentially expand the set of data types LIVS can handle to real numbers, or objects.

Working on these tasks means reading the relevant literature on SyGuS solvers and solver-aided programming, and grappling with the opaque, black-box nature of existing solvers in a way which will provide me with a more intuitive understanding of how they are best used. I will likely have to grow my understanding of various forms of static analysis and complicated program transformations in service of this goal, which will broaden my PL horizons. On top of all of this, working on LIVS is a demanding exercise in software engineering (in a language with which I have limited comfort).

In terms of scheduling, we have agreed on a meeting time of 4:00pm each Monday for two hours. In the remainder of each week I will spend some hours (probably 5-6 on average) working on the above tasks, roughly in the order that they are listed (subject to change, of course). Some of these tasks, such as writing benchmarks, will make up less than the work of a full week, while other more open-ended tasks may take several weeks of work or end up not entirely resolved, as we shift our goalposts and learn more about the challenges and solutions inherent in each problem by tackling it.
The deliverables for this project obviously include the codebase itself – importantly however, the PLDI ‘20 deadline conveniently comes near the end of the semester, and we will prepare and submit a paper that will serve as a summary of LIVS and a written report on the work we did. I will also create a website as mentioned in the CPSC 490 requirements.

As I am planning to apply to PhD programs at the end of the calendar year, preparing this paper will be a demonstrative experience as well as a boon to my applications. I expect that my future research will be in PL and the knowledge I take from working on LIVS will (in addition to being personally exciting) be a springboard for delving into further research topics.