Angel Cui and Gregory Schare
COMS 3997 BC
Seminar in Program Synthesis
Mark Santolucito

Review: *Semantics-Guided Synthesis*

Evaluation

1) A score for your overall evaluation - either accept or reject. Usually, review platforms will also ask for your confidence on a scale of Expert or Knowledgable (there are other options, but you should always be one of these).

We accept the paper. It provides a substantial novel contribution that fills a need in the program synthesis community and opens the door for plenty of future work. Though we are novices in program synthesis and unfamiliar with Constrained Horn Clauses and regular-tree grammars, we are able to appreciate the SemGuS framework at a high level.

Summary

2) A summary of the work, accessible to those in the class. This should highlight the major contributions of the work and how it fits into the related work cited in the paper. If you are able, try to also connect the work to topics we have covered in class

SemGuS attempts to fill a gap between existing program synthesis tools. Solver-aided languages like Rosette and Sketch are language-specific and cannot deal with infinite search spaces, and SyGuS, the language-agnostic way of expressing synthesis problems, does not support semantics outside of supported background theories.

SemGuS bridges the gap by providing a SyGuS-like framework for specifying synthesis problems with arbitrary semantics. It is capable of handling problems involving infinite search spaces and unbounded loops, as well as proving unrealizability of imperative synthesis problems. You can think of SemGuS like an intermediate representation for synthesis: it provides a universal way to express problems coming from a variety of semantic contexts, to be used in a variety of solvers or solver-aided languages. Instead of requiring N^2 custom symbolic compilers for N semantics and N solvers, you only need to write N SemGuS instantiations.

SemGuS enables inputting custom semantics by allowing the user to provide Constrained Horn Clauses describing the semantics of the synthesis problem. Constrained Horn Clauses can be solved by some off-the-shelf constraint solvers like z3. Additionally, Constrained Horn

Clauses can be rewritten into inference rules, allowing a more user-friendly way to input the semantics. The user must also provide, as in SyGuS, a grammar specifying the syntax. The syntax is encoded as a regular-tree grammar.

The solving procedure involves invoking a constraint solver on the problem, which looks like attempting to complete a proof tree of the query. If a proof can be found, then the term used to complete the proof is the synthesized program. If no proof can be found, then the problem is unrealizable. The query's preconditions are that the resulting term is syntactically valid and that the semantics are correct for all input/output examples. Though the solving procedure operates on input/output examples, SemGuS can take specifications as either logical formulae or sets of input/output examples by using CEGIS to convert logical formulae into examples.

As mentioned above, SemGuS is able to prove unrealizability. This is its other main contribution (besides the ability to handle custom semantics). Rosette and Sketch are unable to do this. SemGuS is actually more geared towards proving unrealizability than synthesizing programs, but this is not prescribed; improvements to the synthesis procedures would make it efficient for both sides. Unrealizability can be made efficient by optimizing *lemma discovery*–that is, finding strong lemmas that prevent progress in the proof tree as early as possible. Lemma reuse and vectorized semantics are another optimization.

The authors provide a simple implementation of SemGuS, called MESSY. They evaluate MESSY's performance against other solvers on synthesis and realizability (where applicable) of SyGuS benchmarks, imperative programming problems, and regular expressions. Overall, MESSY performs well on realizability and modestly on synthesis. MESSY's underwhelming performance on synthesis is reasonable considering the other tools have had years of development and optimization; it performs on-par with early SyGuS solvers, indicating its infancy and potential for improvement. Since most other tools are unable to prove unrealizability or handle imperative programs, SemGuS's advantage is its ability to discover lemmas for proving unrealizability and the flexibility of its semantics.

The paper describes some optimizations that improve SemGuS's runtime. These include: rewriting the problem as a tree, list, or array; fused semantics; vectorizing the input/output examples so that multiple examples can be checked at once and work can be reused rather than redundantly duplicated; and using abstract or underapproximating semantics when only one-sided analysis is necessary. MESSY implements all of these optimizations and the evaluations found that SemGuS is most efficient when a combination of vectorized and fused semantics are used.

Future work includes improving SemGuS's lemma discovery, abstract semantics, and building dedicated solvers specialized to SemGuS problems.

Critical Review

3) A critical review of the work. What evidence are you using to support your overall evaluation score? Are there issues with the work/writing/evaluation? What stands out as truly impactful about this work? What excites you about this work?

We acknowledge the impactful nature of this paper. Its contributions are genuinely novel and provide previously-unseen flexibility in defining synthesis problems. No other existing tool (as far as we and the authors are aware) provides similar functionality to SemGuS. In the growing field of synthesis and solver-aided languages, an "IR for synthesis" is much needed; SemGuS fulfills this need. It opens the door for plenty of future work in improving or modifying the framework, optimizing MESSY or creating new implementations, and allowing other similar systems to take inspiration from its ideas.

Additionally, we applaud the writing and style of the paper and its expository clarity. As readers who are new to synthesis and totally unfamiliar with many of the theories and technologies employed in SemGuS, we appreciated the explanations of necessary background knowledge and the thoughtful, natural structure. As readers in general, we especially appreciated the clarity of the paper; by explaining its contributions and functionality at various levels of abstraction throughout the various sections, we were able to understand the framework first at a high level and then in technical detail. This made the paper very approachable and it means it has the potential to have great impact.

That being said, we take issue with the length. At 32 pages, we wonder if the authors may have gone somewhat overboard in their explanations. We felt that some aspects, like the discussion of unrealizability, was repetitive because of how frequently it was re-explained throughout the paper. Since the authors have already provided accessible resources in the form of a 5-minute talk and a vision paper, we would have liked it if the paper were slightly shorter and more straight-to-the-point. Alternatively, we would appreciate a short, technically-dense version of the paper designed to explain the technical aspects without as much overview.

Finally, we express concerns and uncertainty about the usability of the framework. The research included no user study, making it difficult to assess whether this design is ergonomic and effective in a practical setting where a researcher or programmer wants to use it to design a new synthesizer for a domain-specific problem. MESSY is a simple implementation that was created for the purpose of initial evaluation; it is not designed for usability. As a caveat, we acknowledge that this paper is mainly a work of developing a theoretical framework and, as such, a user study belongs in future research, but we contest that the "Future Work" section did not mention a possible future user study. Despite this, the authors assert that "given a library of

various semantics for operators, defining a new SemGuS problem is just as easy as defining a SyGuS problem." We hope that future work will explore the meaning of "easy" here and identify ways in which the design of the framework can be improved.

Citation

Jinwoo Kim, Qinheping Hu, Loris D'Antoni, and Thomas Reps. 2021. Semantics-Guided Synthesis. *Proc. ACM Program. Lang.* 5, POPL, Article 30 (January 2021), 32 pages. https://doi.org/10.1145/3434311