

Technical Perspective

Program Synthesis Using Stochastic Techniques

By Sumit Gulwani

PROGRAM SYNTHESIS INVOLVES discovering a program from an underlying space of programs that satisfies a given specification using some search technique.³ It has many applications including algorithm discovery, optimized implementations, programming assistance,⁵ and synthesis of small scripts to automate repetitive tasks for end users.⁴ Its success relies heavily on efficient search algorithms to navigate the underlying huge state space of programs. The authors of the following paper have developed a stochastic search technique and applied it to program optimization. The impressive results of their implementation STOKE on hard program optimization benchmarks illustrate the promising potential of stochastic search to hard program synthesis problems.

The specification for program synthesis can be in the form of a logical declarative relationship between inputs and outputs. Examples or demonstration traces are a popular specification choice for end-user programming.⁴ In program optimization, when viewed as a synthesis problem, the specification consists of inefficient programs that need to be translated into functionally equivalent but more efficient programs.

Multiple solutions may satisfy the Boolean constraints in the specification. In such cases, preferences can be specified using an optimization function. In programming-by-examples, where the number of solutions may be several powers of 10, ranking functions over program features are used to guess an intended program.⁴ In program optimization, the goal is to prefer programs with smaller runtimes. STOKE's use of sum of average latencies of the involved instructions serves as a good static approximation to the intended measure.

The search space in program synthesis requires a trade-off: expressive


enough to describe programs of interest, while restricted enough to allow efficient synthesis. Various domain-specific languages have been designed for synthesis purposes^{1,3} that meet this trade-off. In program optimization, a common choice is loop-free instruction sequences of bounded length. While prior techniques restrict the space to 10–15 opcodes or require specifying a small set of relevant opcodes for a given problem instance, STOKE significantly advances the state of the art by allowing nearly 400 x86-64 opcodes.

Search Technique

A simple search strategy is to enumerate programs in the underlying space in order of increasing size. However, this does not scale to huge search spaces of the kind considered by STOKE. Another strategy is to reduce the (second-order) search problem to (first-order) constraint solving³ and leverage off-the-shelf SAT/SMT solvers like Z3.² This allows building over huge engineering advances made in SAT/SMT solving, but does not allow effectively incorporating optimization constraints. Version-space algebra-based techniques⁴ incorporate preferences by computing the set of all/many solutions in a first phase, and then selecting the highest-ranked solution in a second phase. STOKE also leverages a two-phased approach. Its first phase finds algorithmically distinct solutions, while the second phase finds efficient implementations of code sequences discovered by the first phase.

Stochastic search. STOKE uses stochastic search for each of its two phases. This includes an appropriately defined cost metric, and MCMC sampling to select a next candidate. The first-phase cost metric is based on functional equivalence to the target input sequence (a Boolean constraint). The second-phase cost met-

ric combines functional equivalence measure with a performance metric (an optimization constraint). In order to define a smooth cost metric over Boolean program equivalent constraints, STOKE uses two clever heuristics: use of Hamming distance to measure closeness of generated bit-values to the target on a representative test input set, and rewarding generation of (almost) correct values in incorrect locations.

Interdisciplinary inspiration. STOKE combines techniques from software engineering, programming languages, and numerical optimization. It uses test input generation (Intel's PinTool) for generating representative test inputs for evaluating equivalence cost metrics during MCMC sampling. It uses automated theorem proving (Microsoft's Z3) for verifying equivalence of the synthesized sequence in a post-processing step. Recent extensions that search over loopy program spaces leverage invariant inference techniques for verifying equivalence. STOKE is a great exercise in interdisciplinary inspiration for efficient search algorithms for hard synthesis problems. This is timely and significant, given recent renewed interest and promising developments in the area of program synthesis across various communities. 

References

1. Alur, R. et al. Syntax-guided synthesis. In *Proceedings of 2013 FMCAD*.
2. Bjørner, N. Taking satisfiability to next level with Z3. In *Proceedings of 2012 IJCAR*.
3. Gulwani, S. Dimensions in program synthesis. In *Proceedings of 2010 PPDP*.
4. Gulwani, A., Harris, W., and Singh, R. Spreadsheet data manipulation using examples. *Commun. ACM*, (2012).
5. Solar-Lezama, A. Program Synthesis by Sketching. Ph.D. thesis, UC Berkeley, 2008.

Sumit Gulwani (sumitg@microsoft.com) is research manager and principal researcher at Microsoft Corp., Redmond, WA.