A new approach to building solvers

Rohit Singh and Armando Solar-Lezama

Massachusetts Institute of Technology
{rohitsingh,asolar}@csail.mit.edu

1 Introduction

General solvers are the workhorse of a large number of fields. Despite targeting NP complete problems or worse, they can handle problems with millions of constraints and variables making them a major tool in certain fields like automated analysis of software and hardware. In fact, these solvers are so efficient that they are used as black boxes.

But the age of the general purpose solver cannot go on forever because at the end of the day, these are hard problems and there are no general algorithms that will work efficiently for all problems. Even today, the promise of the fully declarative black-box solver is not entirely accurate; different representations of the same problem have a huge impact in performance, and tools that need maximum performance spend significant effort making sure their constraints are in a form that can be solved efficiently.

Our goal in this work is to stay true (as much as possible) to the promise of a truly declarative universal interface to the solver that allows us to express problems from a variety of domains without worrying about how they will get solved, but get the benefits of custom made solver built for the purpose of one application with no regard for maintainability or other software engineering concerns.

To this end, we present a framework for augmenting the general purpose solver with a way to incorporate domain specific knowledge. This will enable the solver to do things internally that were being done externally by these tools.

2 Motivation: Challenges in building solver interfaces

For building a typical interface to an existing black box solver, a tool developer has to make a lot of choices. These decisions are usally finalized over a span of a few years before the tool can be used for large scale problems. This process is quite tedious and involves tradeoffs between software efficiency and maintainability. Some examples of tools employing similar interfaces are Sketch [6], Jeeves [7], and BBR [3].

3 Auto-Generated Solvers

We believe that by leveraging existing Program Synthesis[5][4] and Machine Learning/Autotuning[1] techniques along with some domain specific knowledge, we can automatically generate parts of the solver that are specialized for that domain. To be more specific, we list the parts that will be automatically generated:

- **Internal Representation Parser:** Choices for basic entities in the internal representation can be obtained by learning statistically significant patterns[2] from some training benchmarks from the domain.
- **Internal Representation Optimizer:** We can obtain rewrite rules using template based program synthesis techniques. Finding an optimal subset of these rules, reduces to an auto-tuning problem. We can also generate efficient code for the optimizer employing tecnhiques to make a tradeoff for maintainability (since nobody would ever have to code these again) against efficiency of the implementation.

– **Encoder to the black box solver:** These rules rely heavily on expert knowledge, so we can let the tool developer provide some choices for different entities in the internal representation or learn some of these from the effort spent on existing tools built on top of similar black box solvers, and then auto-tune to make the best choices.

3.1 Proof of concept: Sketch tool

We've successfully auto-generated a domain specific optimizer/rewrite rules for the Sketch synthesis tool (using the tool itself for synthesizing the optimizer). The auto-generated optimization layer can replace the existing hand-crafted rules that were built over a span of multiple years, and still perform much better than the baseline with minimal rewrite rules.

3.2 For existing tools: Auto-Generated External Optimizers

We can employ the same techniques to build an interface for SMT problems from a specific domain, that can act as a domain specific optimization layer feeding into the existing tool. This establishes compatibility with existing solvers and preserves the expert knowledge that has been built into them already while improving them to fill the gap in their optimality.

4 Conclusion

We argued that building a new solver on top of a black-box solver is a tedious, time consuming process. And, given the state of the art Program Synthesis and Auto-tuning techniques we are confident that we can take on the challenge of automatically generating most parts of these solver interfaces while helping the tool developer focus on only the important and intuitive aspects of the tool. This is the underlying premise of Program Synthesis that enables developers to think only about what matters the most instead of intricacies of the implementation, and we believe that its time we incorporate that into the way we build new solvers.

References

1. Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *International Conference on Parallel Architectures and Compilation Techniques*, Edmonton, Canada, August 2014.
2. Akhil Arora, Mayank Sachan, and Arnab Bhattacharya. Mining statistically significant connected subgraphs in vertex labeled graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1003–1014, New York, NY, USA, 2014. ACM.
3. Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. Partial replay of long-running applications. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 135–145, New York, NY, USA, 2011. ACM.
4. Sumit Gulwani. Dimensions in program synthesis. In Temur Kutsia, Wolfgang Schreiner, and Maribel Fernández, editors, *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 13–24. ACM, 2010.
5. Armando Solar Lezama. *Program Synthesis By Sketching.* PhD thesis, EECS Department, University of California, Berkeley, Dec 2008.
6. Armando Solar-Lezama. The sketching approach to program synthesis. In Zhenjiang Hu, editor, *APLAS*, volume 5904 of *Lecture Notes in Computer Science*, pages 4–13. Springer, 2009.
7. Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '12, pages 85–96, New York, NY, USA, 2012. ACM.