Z3 and SMT in Industrial R&D

Nikolaj Bjørner

Microsoft Research
nbjorner@microsoft.com

Abstract. Theorem proving has a proud history of elite academic pursuit and select industrial use. Impact, when predicated on acquiring the internals of a formalism or proof environment, is gated on skilled and idealistic users. In the case of automatic theorem provers known as Satisfiability Modulo Theories, SMT, solvers, the barrier of entry is shifted to tool builders and their domains. SMT solvers typically provide convenient support for domains that are prolific in software engineering and have in the past decade found widespread use cases in both academia and industry. We describe some of the background developing the Z3 solver, the factors that have played an important role in shaping its use, and an outlook on further development and use.

1 Introduction

SMT has been pursued for decades by logicians and in industrial laboratories. Decision procedures for selected logical theories have been studied by Presburger, who gave a decision procedure for quantified formulas over linear integer arithmetic, postulated by Hilbert in his famous 10th (unsolvable) problem on Diophantine equations to give two examples. They have been integrated with theorem provers NQTHM, Standford Pascal Verifier, EHDM starting in the 1970's and promoted in the context of PVS and Simplify in the 1990s. A set of important confluences materialized in the past 10-20 years for SMT: our understanding of efficient SAT solving advanced with conflict directed clause learning [5] and efficient data-structures [4], "killer" applications, such as dynamic symbolic execution [3], emerged; and a community around SMT benchmarks and formats formed thanks to initiatives by persistent stake-holders [1].

In the following we describe a set of *driving scenarios* that have shaped our development and use of Z3[1], the importance of the SMT *community efforts* to drive usage, the role of *engineering APIs* and *open sourcing*, and conclude by describing some current efforts on applying Z3 in industrial contexts.

2 Driving Scenarios and Research Synergy

Z3 had the fortune to be nurtured in an environment populated with researchers with synergistic pursuits. These initial driving scenarios and synergies with close

[1] https://github.com/Z3Prover/z3

collaborators are significant enablers. Fortunately, the case for SMT and Z3 has broadened over time. In 2005 Dynamic Symbolic Execution introduced a sweet spot between fuzzing and model checking by applying symbolic solving to path conditions and coped with partial modeling of system calls or other unmodeled instructions using concrete run-time values. It facilitates an established part of software engineering, unit tests, exemplified by Pex, and enhances security fuzzing, exemplified by SAGE. Program verification and contract checking were established since the 90's using the Simplify SMT solver. Simplify had hit a performance barrier in its techniques for quantifier instantiation and our first advance with Z3 was to introduce efficient data-structures that would perform simultaneous E-matching on sets of terms and quantifiers [2]. Microsoft Research was also an incubator to the SLAM symbolic model checker, which had instigated the previous generations of SMT solvers at Microsoft: Zapatho solved integer difference logic and uninterpreted functions, Zap2 (dropping "atho") extended the scope to full linear arithmetic, uninterpreted functions, arrays and quantifiers, and Leonardo de Moura and I created a v. 3 from scratch, Z3, dropping "ap".

Further developments in Z3 and SMT solvers generally continue to be based on inspiration from driving scenarios, by improving their existing uses of constraint solving and enabling new uses through a combination of improved solvers and more expressive functionality. In the context of Z3, Christoph Wintersteiger added solvers for machine arithmetic with IEEE floating point theories. We added additional theories, such as for sequences and strings, plugins for adding custom theories, powerful quantifier instantiation engines that act as decision procedures for several quantified decidable theories, specialized solvers for a class of formulas characterized as constrained Horn clauses that serve as a logical layer for symbolic model checking of procedural languages, scalable linear programming by Lev Nachmanson, and optimization features for the case users need to retrieve models that optimize objectives. I like to characterize a common thread in these developments as one throws a new "toy" in the basket and typically smart minds put it to creative and useful uses. As a logical toolbox Z3 enjoys a cross-cut of application areas that go beyond initial uses. Conversely, users constantly put a growing feature set into increased stress-test, which helps raise the quality bar and inspire areas for further innovation.

Z3 had the benefit of an organization that invests in research tooling. While this is not similar to how products are managed, which include service level agreements and support, it allows for a longer term view compared to academic environments where students expire after a few years, or industrial environments that are driven by short term deliverables and therefore require leveraging existing tools.

3 SMT-LIB - a Research Community

The value of the academic initiative in the SMT community cannot be underestimated. It has produced a standard, SMT-LIB [1], which serves as a well designed and documented basis for community efforts. It has produced a large

set of shared benchmarks from industrial and experimental use cases. The barrier of entry of using SMT tools has been reduced, perhaps at expense of the entry point for producing new solvers that can supersede previous solvers. A clear indication of success for the SMT-LIB efforts is that tools that use SMT solvers can cherry-pick solvers in a portfolio, as done in software model checking tools.

4 Tooling and Infrastructure

A fruit from the interaction with Jakob Lichtenberg from the SLAM/SDV model checker team, was an initial API for Z3, exposed as bindings from C, and on top of that with wrappers for OCaml and .NET. It enabled an initial direct integration, even though maintaining a text pipeline (Z3 originally supported a text-based front-end for the Simplify format) is easier to maintain and debug. A very significant development was the addition of Python based bindings. This enabled easy prototyping through high-level, intuitive, scripting. Together with the well-designed SMT-LIB2 text format, these accessible interfaces are possibly the most important enablers for SMT technologies. In comparison, SAT solvers use lower level formats where formulas are already converted to CNF, and variable names are replaced by numerals. Writing a parser for a SAT solver is trivial, but the barrier of entry of using a SAT solver then includes converting formulas to CNF, and tracking variable names as a separate process.

4.1 Development

A common question is: "who are you managing to develop Z3?". Perhaps this was inspired from institutions where professors are project managers, but not developers. While Z3 is over 300KLOC most was written by relatively few contributors, and development is synergistic with evaluating experimental research questions: e.g., develop more efficient decision procedures. Check-ins into GitHub are monitored by two services, Travis and VS-build, that compile to several target platforms and run unit tests. We use a couple of thousand Azure compute nodes to run full SMT-LIB regressions on check-ins. A custom distributed test infrastructure performs file-sweeping and presents SMT friendly output.

4.2 Open Sourcing

Z3 was open sourced in two stages. In the fist stage, the code was shared and open for academics to use and modify for any research purpose. In the second stage, Z3 was open sourced under an MIT license and moved to GitHub which opened up for significant traction, especially contributions around improved interfacing and later on solving internals. It eradicated several barriers for commercial uses, such as the one a researcher faced when he wanted to acquire Z3 for his company and went through a futile email thread of 7 different parties and 20+ emails because Z3 was only available from the Microsoft online store and not through re-sellers. The open license terms in place today mean that Z3 is integrated in commercial

products without royalties. Open sourcing applies to a dominant number of other tools from Microsoft Research today and overall fits well into a modern era where code development is eased by online tools; and an environment where research code is shared as part of advancing science, advancing usage, and resulting in feedback and improvements.

5 Push, Pull and Confluences

Industrial uses of symbolic solving is a combination of push, pull, and confluences. Program verification has been mainly pushed as a scientific, academic pursuit of ideal software development, but thanks to an active community and tools it is taking inroads with systems such as the Verifying C Compiler, VCC, the Microsoft Research Everest project for verifying secure socket layer implementations. Z3 is part of crypto blockchain verification utilities, including Etherum. The LLVM toolsets, and a development version of the Visual Studio compiler, use Z3 for checking correctness of compiler transformations, and it is used for super-optimization of code snippets. Network Verification is an active area in the management of wide-area and corporate networks; with growth and complexity outpacing traditional lower level network management tools, Z3 and symbolic solving have become useful ingredients for managing networking based on *intents* with deployments in Azure, Amazon, startup code bases and in advanced academic prototypes. Broadly speaking, symbolic solving is being embraced as part of systems developments and deployments, perhaps resembling deployments of operations research optimization tools. New confluences are emerging with the software industry's quest for data-driven and learning-based experiences: operations research and SMT solvers add capabilities to integrate solving and optimization capabilities.

References

1. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.
2. Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, pages 183–198, 2007.
3. Patrice Godefroid, Nils Klarlund, and Koushik Sen. DART: directed automated random testing. In *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, pages 213–223, 2005.
4. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.
5. João P. Marques Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999.