Aiutare: A Modular Benchmarking Framework

Lukas Finnbarr O'Callahan
finnbarr.ocallahan@mail.utoronto.ca
University of Toronto

David Tang
itangdav@gmail.com
University of Toronto Schools

1 INTRODUCTION

The process of benchmarking software on numerous queries can produce a large amount of information, such as program execution runtimes and properties of each input query. Effectively using this information to gain new insights into program performance and correctness is challenging. In response to this difficulty, we developed Aiutare: a modular benchmarking framework that runs on a set of user-defined programs and their input benchmark files. Aiutare helps researchers by abstracting the benchmarking and data storage processes, allowing users to easily manipulate output for visualization, plotting, and testing. In this paper, we introduce the framework and demonstrate its flexibility by applying it to find bugs in the solutions generated by popular SMT solvers.

2 DESCRIPTION OF AIUTARE

Aiutare is a Python program outlined in Fig. 1 and detailed below. We chose Python for the project as all University of Toronto computer science students are familiar with the language.

2.1 Inputs

The user provides Aiutare with a file named `config.py` which consists of a dictionary containing the following items:

(1) A set of P **programs** (1.1), provided as a list of filepaths to the program executables, including any command-line arguments.
(2) A set of F **input files** (1.2), provided as a list of filepaths. All P programs provided should be able to run these input files and print output to the console.
(3) A set of P **schemas** (1.3), provided as a list of filepaths to Python files. We provide default schemas to record program runtime and answers, but the user can optionally write a custom schema in the form of a Python function for each program.

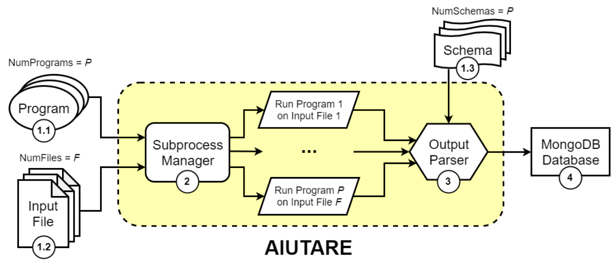

Figure 1: Aiutare framework architecture.

2.2 Subprocess Manager

The Subprocess Manager (2) uses Python's `subprocess` module to run every program on every input file in parallel. The `Popen` interface is used to execute child programs; each invocation consists of a filepath to the program executable (1.1) including any desired command-line arguments, and the filepath of the input file (1.2). A complete invocation should be able to run directly in the terminal and have the format:

```
path/to/program_exe -arg=true test_input_file.txt
```

2.3 Output Parser

After terminating, each child program spawned by the Subprocess Manager (2) has its console output redirected to the Output Parser (3). Here, the appropriate program-specific schema (1.3) provided by the user is called to parse this output text into a dictionary of variables describing the performance and results of the child program. This dictionary is then written to Aiutare's MongoDB database (4) as a MongoEngine schema object [2].

2.4 MongoDB Database

Once the Output Parser (3) finishes writing all results to the MongoDB database (4), Aiutare terminates and prints a summary of the benchmarking to the console. The results of the database can then be easily queried as MongoEngine objects or by using any compatible library or application, such as the MongoDB Compass GUI [1]. MongoDB was chosen as the database for Aiutare because it requires no adherence to fixed schemas or knowledge of SQL.

3 EVALUATION

We use a domain-specific instantiation of Aiutare to evaluate its effectiveness. We aim to answer **RQ1**: given a research question RQ_D in domain D, can Aiutare be adapted to generate meaningful results to aid in answering RQ_D?

We selected satisfiability modulo theories (SMT) solvers as our domain and asked the research question RQ_{SMT}: can we develop a method to systematically validate the solutions produced by SMT solvers in order to catch bugs in these tools?

To the best of our knowledge, no systematic validation of SMT solver solutions exists in the literature, making our bug identification method a useful contribution to the field of SMT solver research.

3.1 Domain: SMT Solvers

SMT solvers take SMT queries as input; a query is in the form of a set of variables and a set of constraints on these variables, as seen in Fig. 2.

The solver outputs the answer *SAT* if all constraints can be satisfied simultaneously, or *UNSAT* otherwise. If the solver answers *SAT*, then it also provides a solution: an SMT query with a concrete value assigned to each variable, as shown in Fig. 3.

SMT queries and SMT solver outputs are also easy to manipulate without modification due to standardization under SMT-LIB [3];

variables	constraints
String x	$y + x = n + m$
String y	$n = ""$
String m	$\text{len}(x) = \text{int}(m)$
String n	

Figure 2: Example SMT Query.

String $x = ""$	$y + x = n + m$
String $y = "0"$	$n = ""$
String $m = "0"$	$\text{len}(x) = \text{int}(m)$
String $n = ""$	

Figure 3: Example SMT Solution.

this consistency allowed us to provide Aiutare with a single schema to handle output from all SMT solvers, simplifying our development process.

3.2 Setup

Our approach to answer RQ_{SMT} consists of two calls to Aiutare and a database parsing procedure.

In the first Aiutare call, the user inputs are:

(1) **programs** = an arbitrary number of SMT solvers.
(2) **input files** = an arbitrary number of SMT queries.
(3) **schemas** = modified Python functions that also write solver-produced solutions to the database every time a solver returns *SAT*.

Next, the second Aiutare call takes as inputs:

(1) **programs** = all SMT solvers.
(2) **input files** = all solver-produced solutions stored in the database after the first Aiutare call.
(3) **schemas** = default schemas to record answers of *SAT*/*UNSAT*.

Once Aiutare has populated the database with these results, the remainder of the approach filters through the data, highlighting each program run where a solver called on an SMT query produced an erroneous answer and/or solution (Fig. 4). One bug type occurs when a solver answers *UNSAT* but a counterexample in the form of a supported solution disproves this answer. Another bug type occurs when a solution is deemed *UNSAT* by one or more solvers.

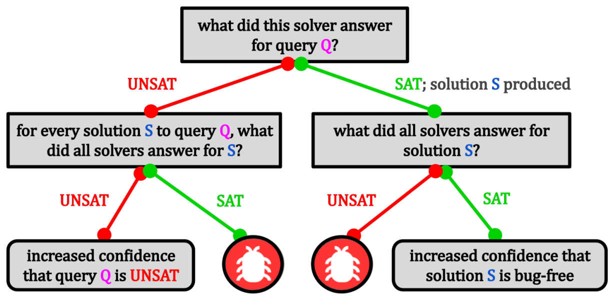
what did this solver answer for query Q?
UNSAT
SAT; solution S produced
for every solution S to query Q, what did all solvers answer for S?
what did all solvers answer for solution S?
UNSAT
SAT
UNSAT
SAT
increased confidence that query Q is UNSAT
increased confidence that solution S is bug-free

Figure 4: Two Types of Bugs.

We selected three versions of Z3 and CVC4 to evaluate:

- **z3_seq** – Z3 using *seq*, the default string solver.
- **z3_str3** – Z3 using *Z3str3* [4], a new alternative string solver.
- **cvc4** – CVC4 using the default string solver [5].

We ran these solvers on 17,936 SMT queries generated by the Kudzu symbolic execution framework [6].

We implemented our approach in less than 200 lines of Python, and ran the configuration of Aiutare located at https://github.com/FinnbarrOC/aiutare on a machine running 64-bit Ubuntu 18.04. Most of the script interacts with well-documented MongoDB APIs, requiring no knowledge of the inner workings of Aiutare. This level of abstraction combined with reliance on common MongoDB libraries ensures that Aiutare is easy to modify and use.

3.3 Results

We were able to systematically find nine bugs: three unsupported solutions and four disproven answers of *UNSAT* with z3_seq, and two unsupported solutions with cvc4.

An example bug taken from the test set of queries is shown in Fig. 5. This subtle bug is hard to catch because z3_seq gave the correct result, *SAT*, but provided the wrong solution.

String $x = ""$	String $x = ""$
String $y = ""$	String $y = "0"$
String $m = ""$	String $m = "0"$
String $n = ""$	String $n = ""$
$y + x = n + m$	$y + x = n + m$
$n = ""$	$n = ""$
~~**len(x) = int(m)**~~	$\text{len}(x) = \text{int}(m)$

Figure 5: Bugged vs. Correct Solution.

Therefore, we positively answer RQ_{SMT} and **RQ1**.

4 SUMMARY AND FUTURE WORK

Overall, we adapted Aiutare to analyze SMT solvers and answer a domain-specific research question, showcasing the flexibility of our framework. Our case study demonstrated that Aiutare can be used to find real-world bugs in current industrial software.

In the future, we plan to apply Aiutare to analyze the runtime performance of SMT and SAT solvers, test verification tools and equivalence checkers, and generate plots and regression models to describe program performance metrics.

ACKNOWLEDGMENTS

We thank Professor Marsha Chechik, Federico Mora, and the UofT Modeling Group for their mentorship and feedback. This research was supported by NSERC.

REFERENCES

[1] [n. d.]. Compass. https://www.mongodb.com/products/compass
[2] [n. d.]. MongoEngine. http://mongoengine.org/
[3] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2017. *The SMT-LIB Standard: Version 2.6.* Technical Report. Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.
[4] M. Berzish, V. Ganesh, and Y. Zheng. 2017. Z3str3: A String Solver with Theory-aware Heuristics. In *2017 Formal Methods in Computer Aided Design (FMCAD)*. 55–59.
[5] Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark Barrett, and Morgan Deters. 2016. An Efficient SMT Solver for String Constraints. *Formal Methods in System Design* 48, 3 (June 2016), 206–234. https://doi.org/10.1007/s10703-016-0247-6
[6] Prateek Saxena, Devdatta Akhawe, Steve Hanna, Feng Mao, Stephen McCamant, and Dawn Song. [n. d.]. A Symbolic Execution Framework for JavaScript.