Hermes: Final Report

Clark Barrett, Mathias Preiner, Yoni Zohar

Stanford University

1 Query Dispatcher

1.1 Description

Part of the approach that underlies the HERMES project is "choosing the right tool for the job". And indeed, sub-problems that are described in a HERMES module are dispatched to various reasoning engines such as SMT-solvers, model checkers, etc. This approach can be leverged also in a lower level of abstraction. When a (sub-)problem that is decided by HERMES to be solved by an SMT-solver, it is still required to determine *which* SMT-solver will be used.

For this purpose, we have implemented and utilized a *query dispatcher*, that makes use of the fact that several solvers can be run in parallel. Thus, given an input file written in the SMT-LIB standard [3], we dispatch it to several solvers in parallel, return the result obtain by the first solver to get an answer, and terminate the runs of the rest of the solvers.

In order to not overuse the system's resources, and given the large number of SMT-solvers, not all possible solvers are executed, but a subset of them is being chosen. To choose this subset efficiently, we need to expect which subset of solvers is best for the particular input problem. The parameter of the input problems that we chose to focus on is the problem's *logic*. Problems in SMT-LIB format are divided into logic, that determine the various types and operators that are allowed to be used. Prominent such logics include, e.g., QF_BV, the quantifier-free logic of bit-vectors, QF_NRA, the quantifier-free logic of non-linear real arithmetic, UFNIA, the quantified logic of non-linear integer arithmetic combined with uninterpreted functions, etc.

Given the input problem's logic (which is typically specified in the first line of the input SMT-LIB file), the way that we project which solvers should be employed is data-driven. For each logic, we choose the solvers that were in the first four places in the yearly SMT competition [9]. The current version of the dispatcher relies on the results from the 2019 competition [1]. In case less than four solvers are supported for the given logic, all of them are chosen.

The dispatcher currently supports the following pool of solvers: CVC4 [2], Z3 [7], Yices [8], VERIT [4], mathsat [6], and SPASS_SATT [5].

In addition to the above functionallity, the dispatcher allows the user to manually choose which solvers will be used, by specifying a list of their names

in the command line.

The following snippet is a trace from running the dispatcher. In each line, there is a timeout of 3 seconds which is enforced on the command that is being executed (using `timeout 3s`). In the first line, CVC4 is specified by the user as the chosen solver, and returns a result. Next, Yices is chosen to run on the same file, and does not return a result. In the next two commands, a different benchmark is used with the opposite results: Yices returns an answer while CVC4 does not. Next, the `-s` option of the tool allows the user to specify more than one solver, and so when both CVC4 and Yices are specified, a result is obtained. Finally, the `normal` keyword of the tool invokes the selection algorithm described above, which succesfully results in a solution.

```
$ timeout 3s python3 dispatcher.py term-mzEEAc.smt2 -s cvc4
sat
$ timeout 3s python3 dispatcher.py term-mzEEAc.smt2 -s yices
$ timeout 3s python3 dispatcher.py term-n2YjEt.smt2 -s cvc4
$ timeout 3s python3 dispatcher.py term-n2YjEt.smt2 -s yices
sat
$ timeout 3s python3 dispatcher.py term-n2YjEt.smt2 -s cvc4 yices
sat
$ timeout 3s python3 dispatcher.py term-n2YjEt.smt2 -s normal
sat
```

1.2 Evaluation

Since the dispatcher chooses the best performing solvers based on the SMT-LIB competition and runs them in parallel, it is expected that it will outperform any single solver, unless it introduces overhead that skews these results. Luckily, the dispatcher is lightweight and performs as expected.

To show this, we compared the performance of the dispatcher with the performance of two mainstream SMT-solvers, namely Z3 and CVC4. As a benchmark set for the experiments we used the benchmarks that were chosen for the 2019 SMT competition in the single-query track on two divisions: QF_NRA and QF_NIA. These divisions include non-linear arithmetic formulas over the reals and integers, respectively. We measured how many benchmarks were solved as time elapsed, with a time limit of 5 minutes and memory limit of 8GB. The experiments were done on a cluster with Intel Xeon CPUE5-2620 CPUs with 2.1GHz and 128GB memory. The results of this evaluation are presented in Figures 1 to 3.

Figure 1 contains two cactus plots. In each plot, each point corresponds to a benchmark, and it is placed on the x-axis according to the time it took to solve the benchmark (or to reach a timeout). The points are ordered according to their corresponding running times on the different solvers, and so two points at the same position according to the x-axis do not necessarily describe the same benchmark. The left plot describes the results on QF_NRA and the right one describes the results on QF_NIA. Consistently, the dispatcher solved more

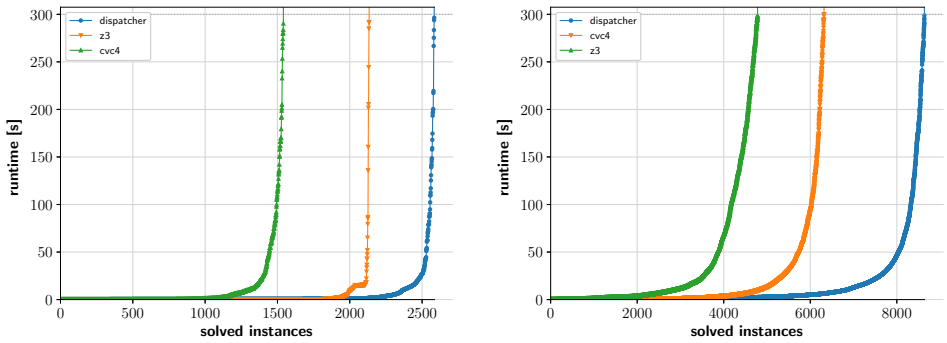

Figure 1: Cactus Plots comparing the dispatcher, Z3 and CVC4 on benchmarks from QF_NRA and QF_NIA.

benchmarks than Z3 and CVC4, overall and also at every time period, for both logics.

Figure 2 contains two scatter plots. In these plots, every point corresponds to a benchmark from the QF_NRA family, and it is placed on the x-axis according to the solving time using CVC4 (on the left) and Z3 (on the right), and on the y-axis according to the solving time using the dispatcher. Thus, points that are on the main diagonal correspond to benchmarks that were solved equally fast by the dispatcher and CVC4/Z3, and points below the main diagonal correspond to benchmarks on which the dispatcher performed faster. As noted in the plots' legends, the color of the points describes by which factor was the better-performing solver faster. It can be clearly seen that most of the points are below the main diagonal on both plots, which means that the dispatcher performed better than CVC4 and Z3 on QF_NRA. Figure 3 presents similar results for QF_NIA formulas.

Overall, out of 2842 QF_NRA benchmarks, the dispatcher solved 2584 in the given time limit, Z3 solved 2134, and CVC4 solved 1542. Out of 11494 QF_NIA benchmarks, the dispatcher solved 8637 in the given time limit, Z3 solved 4782, and CVC4 solved 6320.

References

[1] SMT-COMP 2019 website. https://smt-comp.github.io/2019/.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, CAV'11, pages 171–177. Springer-Verlag, 2011.

[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th*

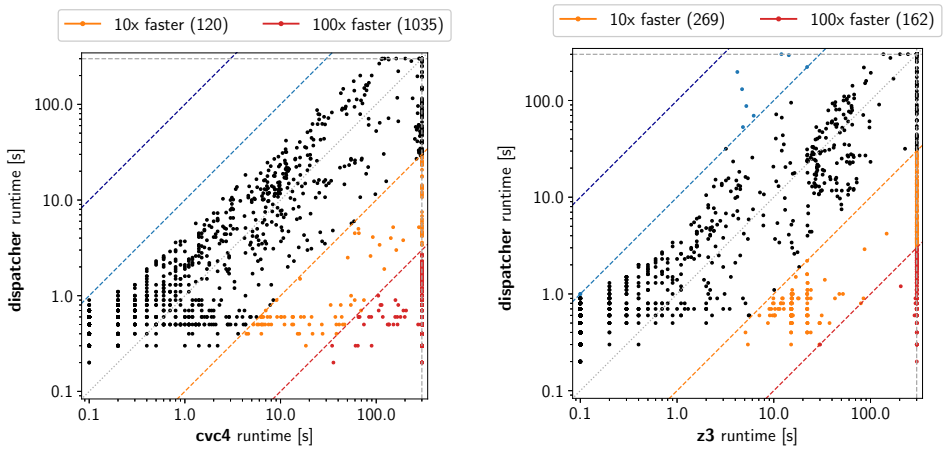

Figure 2: Scatter plots comparing the dispatcher against Z3 and CVC4 on benchmarks from QF_NRA.

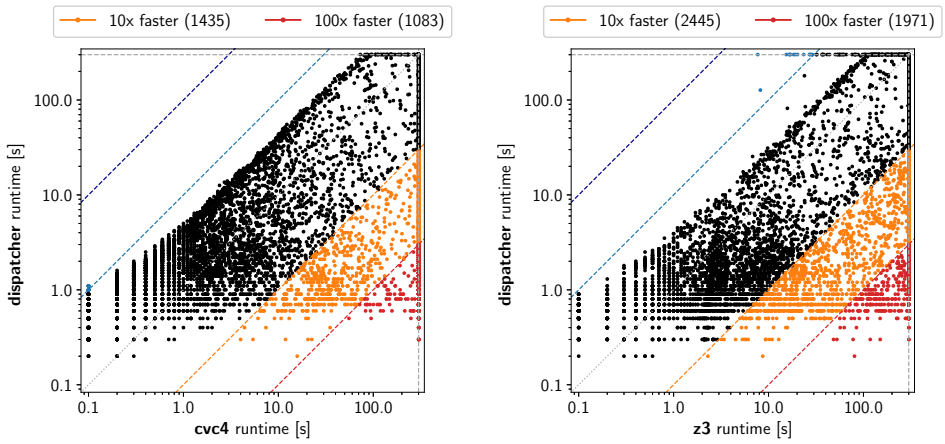

Figure 3: Scatter plots comparing the dispatcher against Z3 and CVC4 on benchmarks from QF_NIA.

International Workshop on Satisfiability Modulo Theories (Edinburgh, UK), 2010.

[4] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. verit: An open, trustable and efficient smt-solver. In Renate A. Schmidt, editor, *Automated Deduction – CADE-22*, pages 151–156, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[5] Martin Bromberger, Mathias Fleury, Simon Schwarz, and Christoph Weidenbach. Spass-satt. In Pascal Fontaine, editor, *Automated Deduction – CADE 27*, pages 111–122, Cham, 2019. Springer International Publishing.

[6] Alessandro Cimatti, Alberto Griggio, Bastiaan Schaafsma, and Roberto Sebastiani. The MathSAT5 SMT Solver. In Nir Piterman and Scott Smolka, editors, *Proceedings of TACAS*, volume 7795 of *LNCS*. Springer, 2013.

[7] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.

[8] Bruno Dutertre. Yices 2.2. In Armin Biere and Roderick Bloem, editors, *Computer-Aided Verification (CAV'2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 737–744. Springer, July 2014.

[9] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. The SMT competition 2015-2018. *J. Satisf. Boolean Model. Comput.*, 11(1):221–259, 2019.