Evaluating Z3's Performance on Real Number Constraints

Empirical Strategies for Tactic Selection and Parallelization

Dimitar Delov[1]
Supervisors: Soham Chakraborty[1]**, Dennis Sprokholt**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Dimitar Delov
Final project course: CSE3000 Research Project
Thesis committee: Soham Chakraborty, Dennis Sprokholt, Andy Zaidman

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Z3 is a widely used SMT solver with support for linear and non-linear arithmetic. While powerful, its performance is dependent on tactics – user-defined strategies that guide the solving process. This paper systematically analyzes the performance of Z3 across various tactic sequences, including ones with parallelization, on benchmarks from SMT-LIB. The results reveal that certain problems can get a speed-up of up to 5 times with the appropriate strategy, however different approaches come with certain limitations. We explore a pattern in linear problems where parallelization and heavy preprocessing make a big difference to performance. Additionally, we discuss two non-linear benchmarks for which it is very evident that the right approach to tactic selection matters and there is no single optimal strategy in SMT solving. These are all important observations since SMT solvers are a key part of many industrial processes where speed and accuracy are crucial.

1 Introduction

Z3 is a theorem prover developed by Microsoft Research [1] that has the capability of asserting that certain conditions can or cannot be satisfied over various theories such as arithmetic, bit-vectors and arrays. An example of a satisfiable instance over the reals ($\mathbb{R}$) is given in Figure 1. We label it as such since we can find values for x, y, z (for example, $x = 1, y = 0.001, z = 2.001$) where the condition is true.

Do there exist $x, y, z \in \mathbb{R}$ such that this is true:
$(x + y + z > 2)$ AND $(10 \cdot y < 0.22)$ AND $(z = 2 \cdot x + y)$

Figure 1: An example of a satisfiable problem over the reals

Z3's impact can be seen in program verification, testing generation and constraint solving in the industry [2, 3]. Despite its widespread adoption, optimizing its performance for real number arithmetic problems remains a significant challenge. The default solving strategies employed by Z3 may not always be suitable for specific problem structures, leading to suboptimal performance in critical industrial applications where solving time directly impacts development cycles and verification processes. Z3 supports user-defined strategies for solving a problem, called tactics. In essence, they are procedures that simplify or attempt to solve a certain goal. However, the selection and configuration of appropriate tactics is non-trivial, as the solver's behavior can vary significantly depending on the mathematical structure of the constraints, the number of variables involved, and the complexity of the relationships between them.

Real number arithmetic problems are particularly challenging because they encompass both linear and non-linear constraints, each requiring different algorithmic approaches. In the context of Z3, we define linear problems as polynomial equations or inequalities of degree at most one, for example: $x + y + 5 = 2$, whereas non-linear problems have polynomials, where the highest degree is at least 2, for instance: $xy + x > 12$. While linear real arithmetic does not usually pose a big challenge to the solver, non-linear problems often require more sophisticated approaches [4]. Additionally, the heterogeneous nature of real-world problems, which often combine both linear and non-linear elements, makes it difficult to predict which tactics will perform best without empirical evaluation.

There is a lot of documentation regarding the theoretical foundations of Satisfiability Modulo Theories (SMT) solving, including the idea behind Microsoft's Z3 [5]. Current lit-

erature suggests adjusting SMT solvers by addressing certain flaws in the design of their algorithms that slow down performance. Such examples are often seen in non-linear arithmetic solving [6, 7]. However, limited research exists on systematic approaches to tactic selection for real number arithmetic problems. While Z3's documentation explains almost all tactics it offers, this is not enough to make an informed decision based on a problem structure. This is because there is a gap in understanding how different tactic combinations perform across different problem families.

This research addresses the primary question: *How do different tactics influence the performance of Z3 in solving real number arithmetic problems?* To systematically address this question, we begin by investigating Z3's internal structure and how it deals with real numbers. Then, the focus of investigation shifts towards tactics, tactic pipelines and parallelization and how the aforementioned can be used to speed up Z3's performance. Finally, we address the limitations of Z3 and suggest how they can be tackled.

Results reveal that for certain problems a specific pipeline can make a notable difference in solving time. We discuss the tactic sequences studied in the research and explain their effectiveness or lack thereof. For linear problems, a pattern is revealed where certain approaches make a notable difference in performance. We also find two non-linear benchmarks that respond very differently to a specific tactic.

This paper is organized as follows. In Section 2, we give a more thorough view of Z3, the SMT-LIB format, and insight on tactics, tactic sequences and the parallelism capabilities of Z3. In Section 3, the methodology we use to examine Z3's behavior and performance is described. Section 4 gives a complete overview of the experimental setup, followed by a comprehensive analysis of the results from it. Section 5 focuses on responsible research and reproducibility of the experiment. In Section 6, we give further interpretations of the results and consider limitations. Finally, Section 7 summarizes the contributions of this paper and gives ideas about future work.

2 Background

This section provides the required background knowledge to understand the rest of the paper. Section 2.1 discusses SAT and SMT solving and Z3 in general. Then, Section 2.2 gives information about real number arithmetic in Z3 and shows examples. After that, we discuss some of Z3's terminology in Section 2.3 and finish off by briefly introducing the SMT-LIB format in Section 2.4.

2.1 SAT and SMT Solvers. Z3

SAT[1] solvers check the satisfiability of propositional logic formulas, containing combinations of Boolean variables [8]. Satisfiability Modulo Theories (SMT) solvers extend those solvers by adding tools for determining the satisfiability of logical formulas with respect to background theories such as arithmetic, bit-vectors, arrays, etc. [9]. Z3 is an open-source implementation of an SMT solver, developed by Microsoft Research, which integrates a DPLL-based SAT solver and various mechanisms for handling background theories [5]. Its architecture is shown in Figure 2.

[1] abbreviation of satisfiability

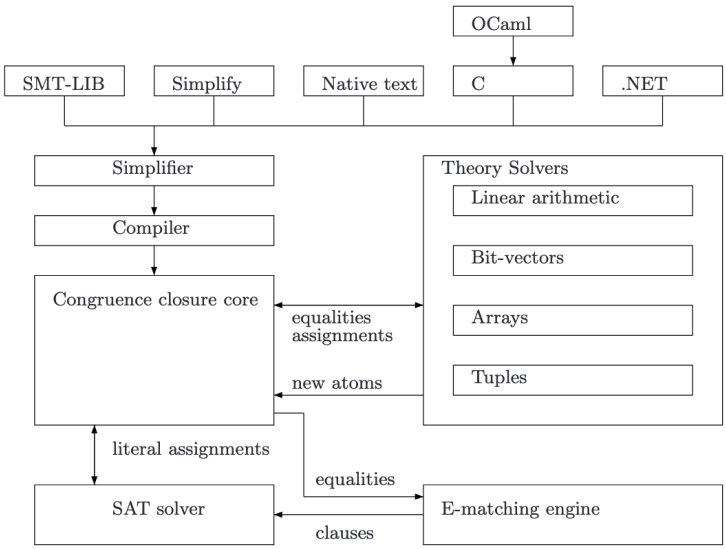

Figure 2: Original architecture diagram of Z3 from [5]

2.2 Real Number Arithmetic in Z3

Z3 supports reasoning over real number arithmetic, including both linear and non-linear **polynomial** constraints, such as the examples in Figure 3a and Figure 3b. While the SMT solver also has support for non-polynomial constraints, they can only be evaluated if there is enough context around them. This means that Z3 cannot solve for unknowns in non-polynomial constraints, but it can substitute their values if they are known, or found at some point of solving. Examples of when the solver cannot determine satisfiability and when it can are respectively given in Figure 3c and Figure 3d.

$$2x + y + z = 2$$

(a) Linear polynomial constraint

$$x^2 + xy + y = 9$$

(b) Non-linear polynomial constraint

$$2^x = 4$$

(c) Undecidable non-polynomial constraint

$$3^x = 9 \text{ AND } x + 1 = 3$$

(d) Decidable non-polynomial constraint

Figure 3: Variants of real number constraints

Furthermore, the theorem prover includes mechanisms for solving problems involving the universal and the existential quantifier. In this research we are focusing on two sets of problems – Quantifier-free linear real arithmetic (denoted: **QF_LRA**) and Quantifier-free non-linear real arithmetic (denoted: **QF_NRA**). By considering only problems without quantifiers, we can better investigate the effects of real-number tactics, since problems in-

volving quantifiers often require tailored approaches revolving around them (e.g. using patterns or elimination) [10].

2.3 Z3 Terminology

The Z3 prover can be used a standalone command-line executable, or through bindings for programming languages such as Python, C#, Java, and others. In both cases the following terms are important for understanding the internals of Z3:

- **Solver**: A core Z3 object that is used to attempt determining the satisfiability of a logical formula. It returns `sat` (satisfiable), `unsat` (unsatisfiable), or `unknown`, which usually indicates a timeout or that some constraint cannot be decided.
- **Tactic**: Tactics in Z3 are used to "guide" the solver how to solve a certain problem – for instance, `propagate-values`. This tactic will substitute known values to simplify calculations. A more specific example is:

$$z = x + 3 \cdot y \text{ AND } x = 2 \quad \xrightarrow{\text{propagate-values}} \quad z = 2 + 3 \cdot y \text{ AND } x = 2$$

- **Tactic Combinator (Tactical)**: Tacticals group together tactics – for example, first eliminate application of unconstrained variables, then propagate values. This is an example of a sequential `then` tactic combinator.
- **Goal**: A set of formulas that when processed by a tactic either yields new sub-goals, indicates satisfiability by returning no new goals, or implies unsatisfiability by returning a new sub-goal: `false`, which can never be asserted.

2.4 The SMT-LIB format

SMT-LIB is a standardized input language for SMT solvers [11]. It defines a syntax for logical formulas and theories. Figure 4 shows an example of a satisfiability problem and its output after running Z3 on it in the SMT-LIB language. Z3 has a built-in parser for the language, treating the parsed tokens as abstract syntax trees that can be transformed into goals.

```
(set-logic QF_LRA)
(declare-const x Real)
(declare-const y Real)
(declare-const z Real)
(assert
  (and
    (> (+ x y z) 2)
    (< (* 10 y) 0.22)
    (= z (+ (* 2 x) y))
  )
)
(check-sat)
(get-model)
```

(a) An example of an SMT-LIB query

```
sat
(
  (define-fun z () Real
    (/ 337.0 250.0))
  (define-fun y () Real
    0.0)
  (define-fun x () Real
    (/ 337.0 500.0))
)
```

(b) Output of running Z3 in the SMT-LIB language

Figure 4: An example of an SMT query and its output after running Z3 on it

3 Methodology

To reason about how different solving strategies influence the performance of Z3, this study uses a **benchmarking**-based approach. This method is specifically appropriate in this case, since it provides systematic, empirical comparisons between the different configurations in a common, controlled environment, allowing for reproducibility of the experiment, as well as pointing out bottlenecks in the combinations of tactics and problems.

3.1 Benchmark Selection

We use benchmarks for real number constraint solving without quantifiers. The complete list of benchmarks is obtained from Zenodo[2] [12] in the form of `smt2` files. In this study we use non-incremental (i.e. with no further constraints added after checking for satisfiability) problems in order to solely focus on the performance of real number operations instead on how the solver manages internal state. The benchmarks are divided into two categories – linear problems (QF_LRA) and non-linear problems (QF_NRA). Both datasets include diverse benchmarks in terms of number of variables and constraints, however they mainly represent industrial problems with the exception of few that are labeled as "crafted".

3.2 Tactic Configuration

The next step after obtaining the benchmarks is to come up with different ways to configure Z3 to test whether tailored approaches such as parallelism or custom tactic sequences speed up Z3's default solver. To do this, we can identify a few common techniques.

3.2.1 Preprocessing strategies

One way to come to a faster solution with Z3 is applying an array of preprocessing tactics to a goal to make it more feasible for the core solver. In this research we consider two levels of preprocessing:

- **Light preprocessing**: `simplify`, `purify-arith`, `propagate-values`, `propagate-ineqs`, `solve-eqs`, `qflra/qfnra`. This approach starts by using Z3's lightweight simplifier, then tries to convert arithmetic operations into just multiplication and addition, propagates values and inequalities[3], solves equations[4] and finally runs Z3's built-in solver for the respective theory. An example of the procedure is shown in Figure 5.
- **Heavy preprocessing**: `simplify`, `purify-arith`, `elim-uncnstr`, `propagate-values`, `propagate-ineqs`, `solve-eqs`, `ctx-solver-simplify`, `qflra/qfnra`. This strategy builds on top of the light preprocessing by introducing elimination of unconstrained variables[5] and applying an expensive context simplifier[6]. In Figure 6 we walk through a more elaborate example.

[2] https://zenodo.org
[3] https://microsoft.github.io/z3guide/docs/strategies/summary#tactic-propagate-ineqs
[4] https://microsoft.github.io/z3guide/docs/strategies/summary#tactic-solve-eqs
[5] https://microsoft.github.io/z3guide/docs/strategies/summary#tactic-elim-uncnstr
[6] https://microsoft.github.io/z3guide/docs/strategies/summary#tactic-ctx-solver-simplify

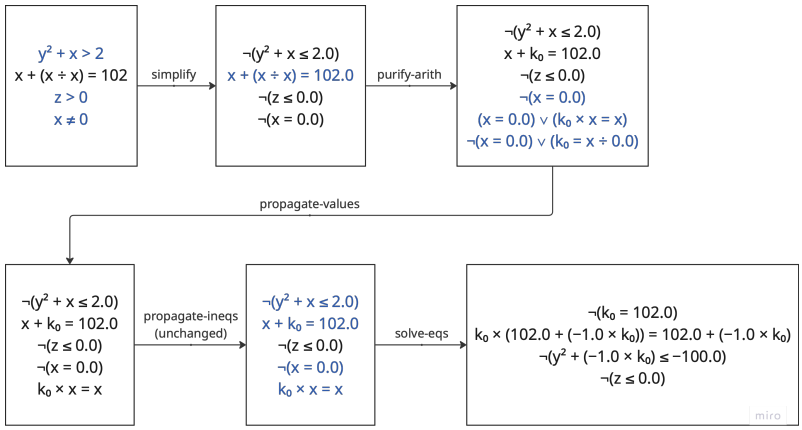

Figure 5: Example of how the light preprocessing pipeline works before handing the simplified problem to the core solver

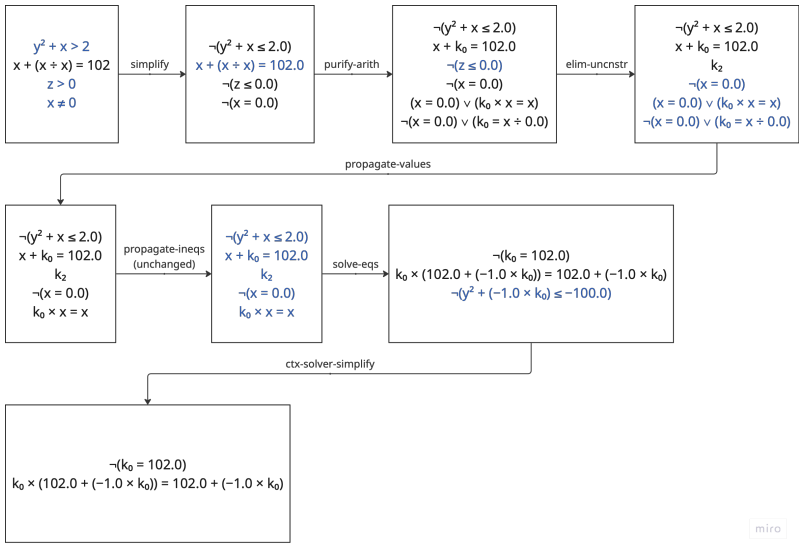

Figure 6: Example of how the heavy preprocessing pipeline works before handing the simplified problem to the core solver

3.2.2 Baseline strategies

We also consider Z3's built-in strategies that directly try to solve problems. For QF_LRA these are `qflra` and `smt`, whereas for QF_NRA we identify three – `qfnra`, `qfnra-nlsat`, and `smt`.

3.2.3 Parallel strategies

We explore two approaches – allowing Z3 to use multiple threads and parallelize using its internal cube and conquer algorithm [9], or by using the tactical "Parallel Or" for the baseline

strategies for the respective problem scope. In the first case, we give Z3 the freedom to use a tactic sequence, or algorithm of its choice on more than one threads. In the second case, we alternate between the strategies from 3.2.2 until one of them solves the problem.

3.2.4 Default strategy

Finally, to see whether the above strategies actually make a difference, we also benchmark the default Z3 solver without tactics or parallelization added to it.

3.3 Evaluation

We conduct the experiment by reporting the time and memory for each combination of problem and strategy. The main analysis is on the time taken, however we also consider memory to show the trade-off between using parallelization and not. With these insights, we identify trends in Z3's performance across different types of problems, given a specific strategy.

For calculating the increase or decrease in the number of problems solved between two strategies, we use the geometric mean of the ratio between the number of solved problems: $(\prod_{i=1}^{n} a_i/b_i)^{1/n}$, where a_i and b_i are respectively the number of solved problems for benchmark i with strategy a and b. The same approach is used for calculating the average speed-up between two strategies, but instead we use the average time of a benchmark. For increases in memory, we use the mean absolute difference: $\frac{1}{n} \cdot \sum_{i=1}^{n} a_i - b_i$, where a_i and b_i are respectively the average memory consumption for benchmark i for strategy a and b in MB.

4 Experimental Setup and Results

In this section we investigate how Z3 performs on different problems using the methodology outlined in the previous section. Section 4.1 gives a concrete view of the environment in which Z3 is used and how benchmarking is configured. In the remaining subsections, we report and analyze the results of 10 QF_LRA and 10 QF_NRA benchmarks[7] with different complexities and sizes. Appendix A gives all 20 aggregated results from the benchmarks.

4.1 Experimental Setup

To ensure that the results are both reproducible and transparent, the code for the experiment has been released in a public repository[8]. The hardware for conducting the experiment was a MacBook Pro, running macOS Sequoia 15.4, with an 8-core Apple M1 Pro chip (2021) and 16 GB of unified memory. Furthermore, the benchmarking suite included Python 3.12.0 in a virtual environment with the following primary packages: `z3-solver` (4.14.1.0), `pandas` (2.2.3) and `matplotlib` (3.10.1). A full list of all libraries and their versions can be found in `requirements.txt`.

The core experiment iterated over a selected benchmark and attempted different strategies to solve each problem in it, reporting time, memory, satisfiability and other statistics into log files. We chose to isolate Z3's work in a subprocess to ensure minimal interference

[7]For the QF_NRA benchmark Strum-MBO we consider only the first 260 problems in an alphabetic ascending order due to resource limitations.

[8]https://github.com/delov23/benchmarking-real-arith-z3

between runs since previous experiments that used separate solvers in the same Python script showed an increase in memory proportional to the number of problems processed, which implied an undesired memory leak. The following tasks were executed sequentially:

1. Define the strategies which we will be testing (outlined in Section 3.2) in a string format in a main script. Get the benchmark folder as input.
2. For every combination of a file name and strategy, run a separate subprocess with a 100-second timeout, which executes the following operations:
   (a) Open the file and parse it into a goal.
   (b) Configure Z3 to solve the goal, using the desired tactic.
   (c) Write the strategy name, time taken, result (`sat`, `unsat`, or `unknown`) and the solver's statistics to `stdout`.
3. If the subprocess succeeds, log SUCCESS and the `stdout`. If it exceeds 100 seconds, we log TIMEOUT and if it fails, log FAIL and print the reason from `stderr` to the console.

After obtaining the log files, the final step was to transform these files into graphs and tables to assess the usefulness of the crafted strategies compared to the default one. With the intuition from the visuals and statistical methods, we investigated whether a certain tactic sequence led to a significant speed-up.

4.2 Collective Strategy Observations

Across the benchmarks, several trends emerged regarding tactic effectiveness. We notice that the lightweight preprocessing strategy often yields similar results to the default one. This is partly because the default implementation of Z3 already applies some simplification and leaves most of the work to the core solver. On the other hand, we can observe that in very few scenarios heavy preprocessing outperforms the default or lightweight strategy. This comes from `ctx-solver-simplify` since it is a very heavy step[9] that can often lead to a timeout.

With regard to the Parallel Or strategy, the first thing we observe are jumps in the maximum memory usage, ranging from 2 times more than the default for most of the linear benchmarks to 10 times for the more computationally-heavy non-linear ones. Particular examples of this are shown in Table 1. Considering only non-timeout instances, the average increase in memory compared to the default strategy is 39.2 MB across QF_LRA benchmarks and – 153.65 MB for QF_NRA ones. Despite the memory consumption, the mean increase ratio in problems solved by the Parallel Or strategy, compared to the default one, is 0.99 for linear problems and 0.92 for non-linear ones, which means that on average it solves respectively 1 and 8 percent less of the considered problems.

4.3 Performance by Benchmark Type

Quantifier-free Linear Real Arithmetic

In general, few of the selected linear benchmarks caused a significant number of timeouts during the experiment, except for **miplib** and **2017-Heizmann**. We observe that for **miplib**, the default strategy, `qflra` and `smt` perform very similarly, yielding the best results

[9] Also pointed out by one of Z3's leading contributors: https://stackoverflow.com/a/11810780

Maximum memory consumption							
Default	Default Parallel	Heavy Pre-pr.	Light Pre-pr.	Parallel Or	qfnra	qfnra-nlsat	smt
hong							
21.97MB	22.78MB	17.93MB	21.76MB	**230.43MB**	21.70MB	28.51MB	17.67MB
kissing							
106.64MB	107.39MB	78.32MB	108.35MB	**200.65MB**	108.22MB	17.32MB	17.68MB

Table 1: Memory consumption across tactics for selected benchmarks

among the others. For **2017-Heizmann**, parallelization and heavy processing make a notable difference, especially in the problem instances: `_count_by_2.i_3_2_2.bpl_11.smt2` (both strategies succeed in under 60 s, whereas the others timeout) and `_array1.i_3_2_2.bpl_11.smt2` (both produce a speed-up of around 2x). The exact times are shown in Table 2.

To investigate the cause of this we make a graph that connects variables to the assertions where they appear. Such graph for the problem `_array1.i_3_2_2.bpl_11.smt2` is given in Figure 7, where variables are on the first and third row (in blue) and assertions are on the second and fourth one (in orange). We note that most of the Heizmann benchmarks follow the displayed pattern. After analyzing it, we distinguish two types of variables – sparsely connected (in the third row) and densely connected (in the first row). Furthermore, we observe that most of the problems, including the ones from Table 2 are unsatisfiable. This means that fast solve time follows from fast contradiction. In that sense, when we have problems like the ones from Heizmann, we want to aim for a more aggressive pruning. From the solver statistics, we can see that the parallel strategy achieves that by exploring more paths and restarting more often to avoid getting stuck in deeper branches. Similarly, heavy preprocessing's context solver simplification invokes solving for specific insertions many times and contradicts faster.

Default	Default Parallel	Heavy Pre-pr.	Light Pre-pr.	Parallel Or	qflra	smt
`Heizmann/_count_by_2.i_3_2_2.bpl_11.smt2`						
TO	51.91s	59.59s	TO	TO	TO	TO
`Heizmann/_array1.i_3_2_2.bpl_11.smt2`						
80.17s	46.62s	53.30s	88.52s	97.34s	98.94	98.46s

Table 2: Selected instances from the Heizmann benchmark

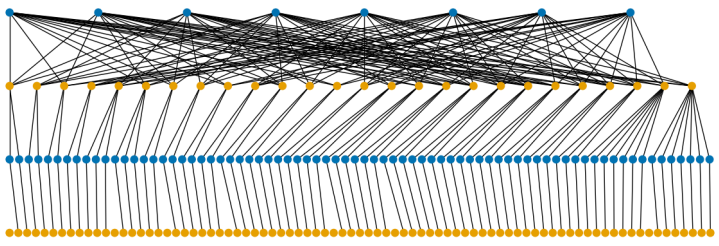

Figure 7: Dependency graph of assertions and variables in `_array1.i_3_2_2.bpl_11.smt2`

Quantifier-free Non-Linear Real Arithmetic

The non-linear benchmarks produce a lot more timeouts with only two suites which have at least one strategy that succeeds among all problems. One observation is that the heavy preprocessing strategy consistently solves fewer problems. For the MGC benchmark, the default strategy solves all 9 problems, whereas heavy preprocessing times out on all instances. For the rest of the benchmarks, the average ratio of problems solved with heavy preprocessing to the default strategy is 0.52, showing that heavy preprocessing solves two times less problems on average. This is likely due to the complex relations in non-linear formulas, which make context simplification infeasible and its use – excessive. In the next subsection we take a deeper look into two specific benchmarks, generated based on [13] that have an application in qualitative analysis of systems of ordinary differential equations.

4.4 Case Study: Sturm-MBO and Sturm-MGC

We begin by exploring the nature of the problems. The problems from MBO contain from 5 – 7 real unknowns, whereas the ones from MGC all contain 9. The benchmarks in MBO can be more formally formulated as:

> Given a non-linear polynomial $P(x_1, x_2, \ldots, x_n)$ of an arbitrary degree where $n \in \mathbb{N}$ and $5 \leq n \leq 7$ (depending on the problem), find variable assignment $x_1, ..., x_n \in \mathbb{R}^+$ such that $P(x_1, x_2, \ldots, x_n) = 0$.

It is also important to mention that the polynomial P is a very long constraint (represented with anywhere from 300,000 to 600,000 characters) as well as that its terms are of very high degrees and are products of multiple variables. Then, we formulate the problems from MGC as:

> Find $\lambda_1 \in \mathbb{R}$ and $\delta, \alpha, \mu, \theta, \gamma_0, vv_1, vv_2, vv_3 \in \mathbb{R}^+$ that satisfy simultaneously one non-linear inequality and four non-linear equations, composed from these variables.

While we have the same complex relations as in MBO, the constraints in MGC are substantially shorter.

We immediately notice that in the first case we try to find the roots of a single equation, whereas in the second one – we have 5 different non-linear constraints. Furthermore, in the MGC benchmarks we have one variable that can also take negative values, unlike in MBO where all values should be positive.

Even though the problems look similar in terms of representation, after running the experiment to see which strategy solves the problems the fastest, it is evident that the two benchmarks behave quite differently. We explore this by looking at two histograms, where each bin shows how many problems were solved in its time interval for two strategies – `smt` (on the left) and the default one (on the right). The bin after 100 s indicates the number of timeouts. Figure 8 shows that MBO problems are solved very efficiently using the `smt` tactic in comparison to the default. The average time it takes `smt` to solve a problem is 12.99 s across 183 non-timeout instances where most problems are solved under 5 seconds. In contrast, the average solving time for the default strategy is 50.47 s across 160 instances.

However, when we have a look at the same histograms for MGC in Figure 9, we see a drastic difference. On average the default tactic solves these problems in just 0.02 s, whereas `smt` solves only one in 47.72 s.

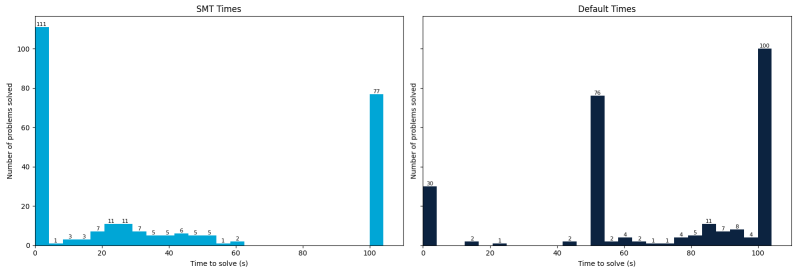

Figure 8: Strum-MBO: SMT Strategy vs Default Strategy

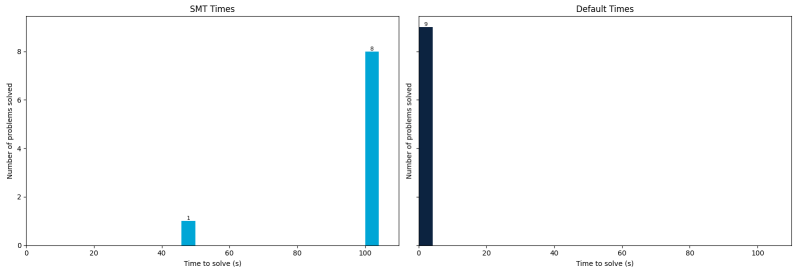

Figure 9: Strum-MGC: SMT Strategy vs Default Strategy

Upon deeper inspection, we also notice that the `smt` tactic only solved problems labeled as unsatisfiable. This can be because this tactic applies a SAT-based SMT solver, which can quickly contradict a part of the single assertion. On the other hand, satisfiable instances require searching a broader space, which only happens with `smt` at the last step. This is namely where we can see how the default use of NLSAT's more advanced reasoning is better suited for the complex relations in non-linear terms [14]. From the two graphs we examined we can see how this intuition holds. MBO has mainly `unsat` problems that involve a single large assertion, so the aim is to quickly find a contradicting variable assignment. With MGC, this is not as trivial since problems involve 5 assertions that must hold at the same time, so we cannot isolate context.

5 Responsible Research

This study adheres to core principles of responsible research, including transparency, reproducibility and integrity. All code produced during this study was made public, including the experiment itself and the data transformation pipelines that analyzed its outcomes. Furthermore, both the raw and the transformed data used in this paper were included in the public repository. In addition, we also considered the reproducibility of the experiment, which was made possible by using a Python virtual environment to list all required packages and their exact versions in the code base. Finally, all experiments were run on hardware that is widely available.

It is also important to acknowledge that benchmarking can often require a large amount of computational resources, which in turn consumes a lot of power from the grid. This can be unsustainable and polluting, depending on the source of energy, therefore experiments

were run only when the results from them had a direct impact on the research. Furthermore, the hardware used was of high power efficiency [15] to ensure power consumption was not needlessly high.

6 Discussion

The observations in this paper are based on 20 selected benchmarks. This is a rather limited selection, which implies that some strategies might still be linked to better performance in other problem structures. However, with the current selection we already see some trends emerging around specific tactic sequences and benchmarks. This means that faster solving does not necessarily have to be linked to new algorithms for SMT solvers, but can instead be brought about by introducing new tactic pipelines.

Furthermore, it is also worth noting that even though we found out that the `smt` tactic can have significant speed-ups for specific unsatisfiable instances, in production settings there are likely going to be multiple situations where we might expect a contradiction, but get satisfiability instead. Therefore, when using the `smt` tactic on problems with a structure similar to MBO, it is important to set a reasonable time limit and explore a different strategy if it expires, otherwise overall performance might become worse.

Finally, we also note that current benchmarks do not consider examples involving more complex mathematical constraints such as modulo operations, which are known to introduce problems[10], or non-polynomial structures. These are core limitations of Z3 and for problems involving such constraints, we need to be extra cautious. For example, when working with modulo operations, aim to minimize their use, and when it is impossible to do so – test small examples in a sandbox environment. For non-polynomial equations and inequalities, it is hard to compose a strategy that deals with them completely. If precision does not have to be infinite, there are tools like dReal[11] that enable the use of non-polynomial functions for producing results with a certain tolerable error.

7 Conclusions and Future Work

In this work, we investigate how we can speed up Z3, a theorem prover, for real number arithmetic problems. More specifically, we show how different tactics influence Z3's performance. In doing so, this research looks into Z3's guiding mechanisms (i.e. tactics and tacticals) when solving problems. We gain empirical insight by benchmarking the prover with different solving strategies on open-source data. Findings show that while linear arithmetic is often quite optimized by default, a problem structure where the majority of variables are loosely connected to assertions might still benefit from a custom approach like using Z3's built-in parallelization capabilities or employing a preprocessing strategy that involves contextual solver simplification. For non-linear problems, we observe that attempting to find the roots of a single polynomial can be contradicted more quickly by relying on the tactic `smt`, instead of the default strategy. However, when we have more complex relations between multiple non-linear assertions, or we expect satisfiability, it is far better to use the default approach.

So far, we have considered Z3 and how it solves real number arithmetic problems. Nevertheless, there exist other SMT solvers with similar capabilities and their own handling

[10] https://github.com/Z3Prover/z3/issues/7464
[11] https://dreal.github.io

of different background theories. Therefore, the idea from this study can be extended by including other provers with the goal to find a match between problem family and a pair of solver and solver strategy. Furthermore, modern programming languages allow for a more intuitive parallelization of programs – for example, Swift's Concurrency model[12], or Python's concurrent futures[13]. Thus, we can consider another mode of parallelization, which is user-defined. For instance, when we encounter a disjunction that does not share context with other assertions, make each term of the disjunction a future (or task) that tries to solve for the term independently. After that, indicate satisfiability when at least one term is satisfiable. If none of them are, return unsatisfiability.

Finally, future work can also be based on a direct application of the findings and data from this research to aid with automatic tactic selection for unseen problems. Current studies already introduce AI algorithms that can decide what strategy to use, based on a goal [16, 17]. One can add to this research by using benchmark scores to fine tune models, or even train them based on the raw data from experiments.

References

[1] Microsoft Corporation. *Introduction*. https://microsoft.github.io/z3guide/docs/logic/intro. Introduction | Online Z3 Guide. Microsoft.

[2] Nikolaj Bjørner. "Z3 and SMT in Industrial R&D". In: *Formal Methods*. Ed. by Klaus Havelund et al. Cham: Springer International Publishing, 2018, pp. 675–678. ISBN: 978-3-319-95582-7.

[3] Agnieszka M. Zbrzezny et al. "SMT Solvers as Efficient Tools for Automatic Time Properties Verification of Security Protocols". In: *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. 2019, pp. 320–327. DOI: 10.1109/PDCAT46702.2019.00065.

[4] Gereon Kremer et al. "Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description)". In: *Proceedings of the International Joint Conference on AutomatedReasoning (IJCAR '22)*. Ed. by Jasmin Blanchette, Laura Kovács, and Dirk Pattinson. Vol. 13385. Lecture Notes in Computer Science. Springer Nature, Aug. 2022, pp. 95–105. DOI: 10.1007/978-3-031-10769-6_7. URL: http://theory.stanford.edu/~barrett/pubs/KRB+22.pdf.

[5] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by David Hutchison et al. Vol. 4963. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. ISBN: 978-3-540-78799-0 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24. URL: http://link.springer.com/10.1007/978-3-540-78800-3_24 (visited on 06/01/2025).

[6] Zhonghan Wang. *clauseSMT: A NLSAT-Based Clause-Level Framework for Satisfiability Modulo Nonlinear Real Arithmetic Theory*. arXiv:2406.02122 [cs]. June 2024. DOI: 10.48550/arXiv.2406.02122. URL: http://arxiv.org/abs/2406.02122 (visited on 06/01/2025).

[12] https://docs.swift.org/swift-book/documentation/the-swift-programming-language/concurrency/

[13] https://docs.python.org/3/library/concurrent.futures.html

[7] Zhonghan Wang. *DNLSAT: A Dynamic Variable Ordering MCSAT Framework for Nonlinear Real Arithmetic*. arXiv:2406.18964 [cs]. June 2024. DOI: 10.48550/arXiv.2406.18964. URL: http://arxiv.org/abs/2406.18964 (visited on 05/30/2025).

[8] Erika Ábrahám, József Kovács, and Anne Remke. "SMT: Something You Must Try". en. In: *Integrated Formal Methods*. Ed. by Paula Herber and Anton Wijs. Vol. 14300. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2024, pp. 3–18. ISBN: 978-3-031-47704-1 978-3-031-47705-8. DOI: 10.1007/978-3-031-47705-8_1. URL: https://link.springer.com/10.1007/978-3-031-47705-8_1 (visited on 06/09/2025).

[9] Nikolaj Bjørner et al. "Programming Z3". en. In: *Engineering Trustworthy Software Systems*. Ed. by Jonathan P. Bowen, Zhiming Liu, and Zili Zhang. Vol. 11430. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 148–201. ISBN: 978-3-030-17600-6 978-3-030-17601-3. DOI: 10.1007/978-3-030-17601-3_4. URL: http://link.springer.com/10.1007/978-3-030-17601-3_4 (visited on 06/02/2025).

[10] Sascha Böhme and Michał Moskal. "Heaps and Data Structures: A Challenge for Automated Provers". In: *Automated Deduction – CADE-23*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 177–191. ISBN: 978-3-642-22437-9 978-3-642-22438-6. DOI: 10.1007/978-3-642-22438-6_15. URL: http://link.springer.com/10.1007/978-3-642-22438-6_15 (visited on 06/02/2025).

[11] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. *The SMT-LIB Standard: Version 2.7*. Tech. rep. Department of Computer Science, The University of Iowa, 2025.

[12] Mathias Preiner et al. *SMT-LIB release 2024 (non-incremental benchmarks)*. Apr. 2024. DOI: 10.5281/zenodo.11061097. URL: https://doi.org/10.5281/zenodo.11061097.

[13] Thomas Sturm and Andreas Weber. "Investigating Generic Methods to Solve Hopf Bifurcation Problems in Algebraic Biology". en. In: *Algebraic Biology*. Ed. by Katsuhisa Horimoto et al. Vol. 5147. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 200–215. ISBN: 978-3-540-85100-4 978-3-540-85101-1. DOI: 10.1007/978-3-540-85101-1_15. URL: http://link.springer.com/10.1007/978-3-540-85101-1_15 (visited on 06/10/2025).

[14] Nikolaj Bjørner and Lev Nachmanson. "Arithmetic Solving in Z3". In: *Computer Aided Verification*. Ed. by Arie Gurfinkel and Vijay Ganesh. Cham: Springer Nature Switzerland, 2024, pp. 26–41. ISBN: 978-3-031-65627-9.

[15] Apple. *Introducing M1 Pro and M1 Max: the most powerful chips Apple has ever built*. https://www.apple.com/il/newsroom/2021/10/introducing-m1-pro-and-m1-max-the-most-powerful-chips-apple-has-ever-built/. Oct. 2021.

[16] Mislav Balunović, Pavol Bielik, and Martin Vechev. "Learning to Solve SMT Formulas". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 10337–10348. URL: http://papers.nips.cc/paper/8233-learning-to-solve-smt-formulas.pdf.

[17] Joseph Scott et al. “Algorithm selection for SMT: MachSMT: machine learning driven algorithm selection for SMT solvers”. en. In: *International Journal on Software Tools for Technology Transfer* 25.2 (Apr. 2023), pp. 219–239. ISSN: 1433-2779, 1433-2787. DOI: 10.1007/s10009-023-00696-0. URL: https://link.springer.com/10.1007/s10009-023-00696-0 (visited on 06/08/2025).

Appendix A Aggregated Benchmark Results

For each benchmark, the first row represents how many problems were solved within a 100-second timeout. The second one shows the average time of successfully solved problems. The third one shows the average maximal memory consumption of successfully solved problems. Table 3 shows the Quantifier-free Linear Real Arithmetic benchmarks and Table 4 shows the Non-Linear ones.

QF_LRA Benchmarks						
Default	Default Parallel	Heavy Pre-pr.	Light Pre-pr.	Parallel Or	qflra	smt
miplib						
25/42 9.80s 31.37MB	24/42 12.63s 41.39MB	20/42 13.69s 50.72MB	24/42 9.98s 52.29MB	23/42 8.55s 75.22MB	25/42 9.75s 30.48MB	25/42 9.82s 30.48MB
keymaera						
22/22 0.0006s 17.41MB	22/22 0.0010s 18.43MB	22/22 0.0007s 17.61MB	22/22 0.0006s 17.21MB	22/22 0.0045s 50.15MB	22/22 0.0006s 17.14MB	22/22 0.0006s 17.14MB
2017-Heizmann						
20/58 24.55s 27.19MB	23/58 33.63s 29.23MB	23/58 31.51s 27.31MB	21/58 26.16s 28.26MB	19/58 25.71s 68.26MB	21/58 31.33s 28.54MB	21/58 31.26s 28.54MB
DTP-Scheduling						
91/91 0.213s 26.30MB	91/91 0.424s 45.06MB	91/91 0.597s 29.58MB	91/91 0.206s 25.57MB	91/91 0.173s 65.44MB	91/91 0.161s 25.02MB	91/91 0.165s 24.81MB
tta_startup						
67/72 4.36s 28.00MB	67/72 3.32s 32.51MB	66/72 4.56s 27.39MB	68/72 4.66s 26.68MB	68/72 5.68s 70.39MB	66/72 3.14s 25.86MB	66/72 3.11s 25.86MB
spider						
42/42 0.0075s 18.65MB	42/42 0.0070s 18.86MB	42/42 0.0064s 18.19MB	42/42 0.0057s 17.97MB	42/42 0.0102s 52.34MB	42/42 0.0054s 18.09MB	42/42 0.0054s 18.09MB
sal_gasburner						
20/20 0.0069s 18.65MB	20/20 0.0062s 18.89MB	20/20 0.0050s 18.18MB	20/20 0.0061s 18.08MB	20/20 0.0101s 52.38MB	20/20 0.0058s 18.14MB	20/20 0.0058s 18.14MB
uart						
70/73 9.69s 30.50MB	69/73 9.05s 36.83MB	68/73 9.07s 30.17MB	70/73 10.34s 30.11MB	70/73 9.20s 75.22MB	69/73 8.67s 29.10MB	69/73 8.77s 29.10MB
clock_synchro						
36/36 5.99s 22.80MB	36/36 9.91s 23.29MB	36/36 9.13s 22.12MB	36/36 6.05s 22.27MB	36/36 6.69s 60.83MB	36/36 6.03s 22.15MB	36/36 6.02s 22.15MB
sc						
144/144 3.64s 27.91MB	129/144 3.93s 39.76MB	129/144 4.68s 27.43MB	144/144 3.32s 27.21MB	144/144 3.82s 70.54MB	144/144 3.40s 27.08MB	144/144 3.56s 27.08MB

Table 3: Benchmark results for QF_LRA

QF_NRA Benchmarks							
Default	Default Parallel	Heavy Pre-pr.	Light Pre-pr.	Parallel Or	qfnra	qfnra-nlsat	smt
hong							
10/20 1.06s 21.97MB	10/20 1.07s 22.78MB	7/20 6.84s 17.93MB	10/20 1.04s 21.76MB	11/20 4.98s 230.43MB	10/20 1.07s 21.70MB	11/20 3.91s 28.51MB	7/20 6.97s 17.67MB
Uncu							
75/75 0.93s 19.03MB	75/75 0.93s 19.84MB	68/75 0.53s 18.53MB	75/75 0.93s 18.92MB	75/75 0.99s 200.48MB	75/75 0.93s 18.85MB	74/75 0.16s 17.40MB	67/75 0.32s 17.88MB
Pine							
241/245 0.56s 18.86MB	241/245 0.56s 19.67MB	194/245 1.33s 19.23MB	242/245 0.82s 18.77MB	225/245 0.41s 199.69MB	241/245 0.50s 18.65MB	243/245 0.43s 17.40MB	187/245 0.34s 17.66MB
Heizmann							
22/69 20.15s 29.52MB	22/69 20.13s 30.21MB	1/69 0.10s 18.88MB	25/69 20.69s 50.52MB	16/69 32.71s 216.69MB	22/69 20.21s 28.90MB	11/69 3.64s 20.69MB	1/69 0.25s 18.78MB
zankl							
114/166 4.92s 65.77MB	114/166 4.98s 66.63MB	78/166 0.39s 19.84MB	116/166 3.88s 115.02MB	109/166 5.16s 231.83MB	116/166 5.91s 68.89MB	96/166 0.22s 18.06MB	85/166 0.17s 18.34MB
Geogebra							
110/112 0.75s 19.61MB	110/112 0.75s 20.42MB	65/112 2.24s 19.60MB	110/112 0.75s 19.58MB	110/112 1.57s 201.81MB	110/112 0.75s 19.49MB	108/112 0.70s 17.68MB	76/112 1.78s 18.25MB
kissing							
18/45 10.60s 106.64MB	18/45 10.53s 107.39MB	13/45 4.99s 78.32MB	17/45 7.94s 108.35MB	17/45 0.09s 200.65MB	17/45 8.06s 108.22MB	13/45 0.00s 17.32MB	17/45 0.03s 17.68MB
Mulligan							
134/135 0.21s 18.85MB	134/135 0.21s 19.65MB	115/135 0.84s 18.86MB	134/135 0.29s 19.07MB	132/135 0.08s 199.56MB	134/135 0.21s 18.73MB	134/135 0.32s 17.56MB	122/135 0.27s 17.59MB
Sturm-MGC							
9/9 0.02s 18.98MB	9/9 0.02s 19.80MB	0/9 N/A N/A	9/9 0.14s 18.95MB	9/9 0.10s 200.46MB	9/9 0.02s 18.66MB	9/9 0.02s 17.41MB	1/9 47.72s 36.69MB
Sturm-MBO							
160/260 50.47s 259.61MB	117/260 37.92s 105.40MB	74/260 10.53s 44.65MB	134/260 45.57s 184.38MB	112/260 0.79s 233.75MB	161/260 50.70s 260.35MB	31/260 1.16s 17.88MB	183/260 12.99s 305.03MB

Table 4: Benchmark results for QF_NRA