CSE 507 Project: Hybrid Numerical & SMT Optimization

Ruizhe Shi[a], Evan Wang[a], Jaedon Rich[a] and lipson[a]

[a] *University of Washington, Seattle, U.S.*

Abstract

Optimizing positive rational functions $\mathbb{R}^n \to \mathbb{R}$ with polynomial inequality constraints using interval arithmetic, affine interval representations, Bernstein polynomial form, and branch-and-bound methods. Our source code is available at https://github.com/srzer/cse-507-project

THIS IS NOT THE FINAL REPORT; WE ARE GOING TO EMAIL ASKING FOR MORE TIME <3 Thank you for being flexible, work life balance does indeed apply, but not for finals? :3

Keywords: interval arithmetic, SMT, Bernstein, affine interval from, dReal

1. Introduction

There are two key elements to introduce: Interval Arithmetic and the dReal SMT solver. [1] Given a collection of variables x_j whose values are each constrained to given intervals $I_j = [a_j, b_j]$, the problem of interval arithmetic is determining the most precise interval to which some arithmetic combination of these variables are constrained. For instance, given x and y , both constrained to the interval $[0, 1]$, we can say that $x + y$ is constrained to the interval $[0, 2]$ and $x - y$ is constrained to the interval $[-1, 1]$. Complications emerge when there are dependencies between our variables; for instance, supposing we have the relation $x = y$, then $x - y$ is actually constrained to the interval $[0, 0]$. We can restate this problem as follows: given a polynomial on a box $f : I_1 \times I_2 \times \cdots \times I_m \to \mathbb{R}$, determine upper and lower bounds on the value of f subject to constraints $g_i(x_1, x_2, ..., x_m) \geq 0$.

There are a number of approaches to interval arithmetic. The goal is to find bounds that are as tight as possible, as efficiently as possible. In general, different approaches have different strengths and weaknesses, and no single method dominates all others across all test cases. Various types of branch-and-prune methods, such as those described below, are used, along with different heuristics for search and subdivision. Some methods are explicitly designed for polynomials, while others can generalize to transcendental functions.[1] Given any set of inequalities involving arithmetic combinations and compositions of polynomials and transcendental functions in several variables (a very broad scope), we are interested in determining whether there are some real assignments of the input variables that will simultaneously satisfy every given inequality. Note equalities can be given as a pair of inequalities, and so are also included. An example problem would be: Is there some real x, y satisfying $\sin(x) + y = e^{xy} \land \tan(x^2 + y^2) \geq \log(x - y)$? These are difficult problems; in fact, generically they are undecidable. This means we cannot simply feed these problems into an SMT solver using nonlinear real arithmetic with transcendental functions.

However, the problem becomes decidable if we take two actions: First, we want to restrict

*

[1] Some algorithms handle transcendental functions by simply using Taylor approximations.

our search to a compact domain, say a box, by putting bounds on every variable individually. Second, we can introduce an error tolerance instead of looking for an exact solution. That is, we can choose any $\delta > 0$, and then for every constraint $f(x) \geq 0$, we weaken this to $f(x) \geq -\delta$, and an equality $f(x) = 0$ will become $|f(x)| \leq \delta$.

2. Overview

If we have a way to determine satisfiability of a set of constraints, we also have a way to perform optimization: To minimize a function $f(x)$ subject to constraints $g_i(x)$ on a given box domain, we can repeatedly check the satisfiability of the system $f(x) \leq C$ together with $g_i(x)$. If this system is satisfiable, we know the optimal solution is at most C , and if it is not satisfiable then the optimal solution is at least C . Therefore if we start with a known lower bound on the value of $f(x)$ and a known feasible solution (to get an upper bound on the minimum), then using a binary search will efficiently converge to the solution of the optimization problem. Using this strategy, dReal can solve optimization problems.

We can hope to make this more efficient by performing this bounding of $f(x)$ simultaneously to our branch-and-prune optimization procedure. Interval arithmetic on $f(x)$ will give us an initial lower bound on $f(x)$. As we branch-and-prune in a depth-first search to find regions satisfying the constraints and eliminate regions that are infeasible, once we find a region where the constraints are δ-satisfiable we can find an upper bound for $f(x)$ in this small region. This gives an upper bound on our optimal solution. From this point forwards, in our branch-and-prune process we have another mechanism for pruning: Using interval arithmetic, we can find a lower bound for $f(x)$ on a given box, and if this lower bound is greater than our current upper bound on the minimum of $f(x)$, then we know $f(x)$ will not be optimized within this box and we can prune the entire box. Eventually we will explore the entire feasible domain and be left with small feasible regions where $f(x)$ is smallest. Having reduced the problem to small boxes, it is then efficient to find the minimum of $f(x)$ on each of these boxes directly using the method described above, as implemented in dReal.

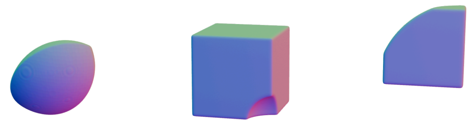


Figure 1: From left to right: hand-written example, Singular Edge, and Rational Bowl optimization search space renders.

Our goal is to see if this approach can outperform dReal's approach to optimization. In particular, we considered rational functions in several variables, and hoped to optimize the procedure for this specific class of functions. In addition to the use of dReal, our approach drew heavily from two previous techniques: Bernstein polynomials and affine arithmetic.

2.0.1. Bernstein Polynomiial Form

Bernstein polynomials give an algorithm for computing exact upper and lower bounds on a given polynomial. It is computationally expensive to compute the initial bounds, however the method of Bernstein polynomials interfaces well with subdividing along boxes so that subsequent refined bounds are more efficient to compute. We apply this method separately on the numerator and denominator of our objective function and then use naive interval arithmetic to bound the resulting ratio, which results in an inexact bound.

2.0.2. Affine Arithmetic

Affine arithmetic is a method for obtaining an approximate interval bound on a polynomial, and there is both a partial/first-order affine arithmetic and a full/higher-order affine arithmetic, the latter being more precise but involving more computation. Bernstein polynomials are very technical, but the method of affine arithmetic can be illustrated simply by example. If our variables are constrained to the intervals $x \in [2, 4]$ and $y[3, 7]$, then we define auxiliary variables constrained to $[-1, 1]$, call them e_1 and e_2 , so that $x = e_1 + 3$ and $y = 2e_2 + 5$. This normalization allows us to handle dependencies; when we compute the interval for $x-$ x we compute $(e_1 + 3) - (e_1 + 3) = 0$. The purpose of normalizing to $[-1, 1]$ is so that products of these auxiliary variables are again variables lying in $[-1, 1]$. For instance, when applying affine arithmetic to the function xy we will get a cross term with $e_1 e_2$. In the partial or first-order affine arithmetic, we would simply substitute in -1 or 1 here when computing extreme values. In higher-order affine arithmetic, these cross terms are replaced with new auxiliary variables, and then again dependencies between different instances of these cross terms can be tracked.

2.1. Our Branch & Bound Algorithm

Our general algorithmic idea is a branch-and-prune approach, which is inspired by dReal: Given the initial box constraint, it will perform interval arithmetic to obtain bounds on the constraint functions to determine whether the constraints are potentially feasible. If not, it returns UNSAT. Otherwise, it subdivides the box into smaller boxes and repeats the procedure on each sub-box: If a sub-box is determined to be infeasible based on the interval arithmetic bounds, it is not explored further, it is pruned. Otherwise, the sub-box is again subdivided into smaller boxes. Interval arithmetic gives imperfect bounds on a function, but subdividing to smaller boxes with tighter interval constraints usually results in more precise bounds, which is the advantage of subdividing. dReal will continue to either prune or subdivide until the box size is so small that the constraint functions vary by less than δ within the small box, at which point the function value is determined on that box within δ precision. Now we present how we improved the vanilla idea through implementing new techniques as below.

2.1.1. Objectives

Our main focus will be rational objective optimization with constraints. The rational objective function $f(x) = p(x)/q(x)$ is represented by two lists of terms: the numerator terms $[[c_i, e_i], ...]$, and the denominator terms $\left[\left[d_j, g_j\right], ...\right]$. For example, if the objective is $f(x, y) = \left(2x^2 y + 3y^2\right)/\left(x^2 + y\right)$, then we have the numerator terms as $[[2, [2, 1]], [3, [0, 2]]]$, and the denominator terms as $[[1, [2, 0]], [1, [0, 1]]]$.

2.1.2. Step 0. Initialization

The algorithm begins with an initial search box that is assumed to contain all feasible solutions and does not include any poles of the rational function. A minimal box size is also defined, usually equal to the numerical tolerance used by dReal (for example, `1e-3`). The global lower bound is initialized, and the initial box is pushed into a queue of boxes to process.

2.1.3. Step 1. Basic pruning

For each box, the algorithm first applies a pruning step using the affine method (we've also tried Bernstein method, which turns out to be efficient in $n = 3$ case, but inefficient for larger dimensions) to estimate a rough lower bound of the objective function on that box. If this rough bound is already worse than the current global lower bound, the box is discarded without further work. Otherwise, the algorithm asks dReal to search for a feasible point inside the box that improves the current lower bound by at least a small threshold. If dReal reports that no such point exists, the box is discarded. If a better point is found, the global lower bound is updated.

2.1.4. Step 2. Handling full feasibility

If the box is already smaller than the minimal box size, it is skipped. Otherwise, the algorithm uses dReal to check whether the entire box is feasible. If it is fully feasible, then we provide two approaches to handle this. The approach 1 is to use dreal's Minimize to directly obtain a lower bound, note that the box is smaller than the whole original box, so it is still efficient; the approach 2 is to invoke an iterative splitting process to obtain a lower bound on this box using affine method.

2.1.5. Step 3. Splitting

If the box is only partially feasible, the algorithm simply splits it into two smaller boxes and continues processing. Beyond simply splitting the longest edge into half, which is the default implementation, we implemented a gradient-guided splitting heuristic, which calculates the gradient direction of the objective to optimize, and then selects the edge which best aligns the direction of the gradient.

3. Algorithms

3.0.1. dReal Algorithm Overview

The dReal solver implements this relaxation to obtain a δ -complete algorithm for determining satisfiability: If there is a solution to system of inequalities in the given bounded domain up to a tolerance of δ , then dReal will find it; if there is no solution, dReal returns UNSAT. Notice that if there is no solution up to a tolerance of δ , then there is also no solution to the exact original problem. The complexity class is PSPACE, but in practice dReal performs very well.

dReal uses a branch-and-prune approach: Given the initial box constraint, it will perform interval arithmetic to obtain bounds on the constraint functions to determine whether the constraints are potentially feasible. If not, it returns UNSAT. Otherwise, it subdivides the box into smaller boxes and repeats the procedure on each sub-box: If a sub-box is determined to be infeasible based on the interval arithmetic bounds, it is not explored further, it is 'pruned'. Otherwise, the sub-box is again subdivided into smaller boxes. Interval arithmetic gives imperfect bounds on a function, but subdividing to smaller boxes with tighter interval constraints usually results in more precise bounds, which is the advantage of subdividing. dReal will continue to either prune or subdivide until the box size is so small that the constraint functions vary by less than δ within the small box, at which point the function value is determined on that box within δ precision.

4. Results

4.1. Summary

Performance differences may derive from the structure of the search space. Our method works well on some problems, due to our specialization on rational functions. On other problems, dReal stood out compared with all other surveyed methods.

On Pole Avoidance in 2: The objective function is $1/(x+y+z-2.5)$ on a simple box. This function is monotonic and smooth over the entire feasible region. The gradient provides a very strong and consistent signal that our branch-and-bound algorithm can use effectively to quickly prune large parts of the space and converge on the minimum without much wasted effort.

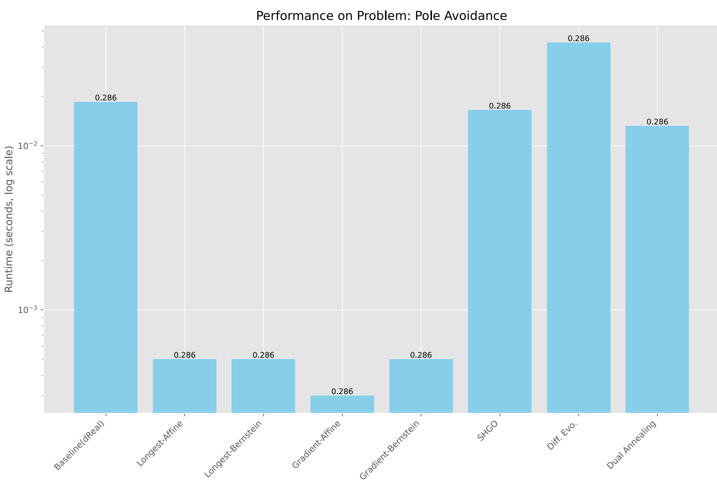
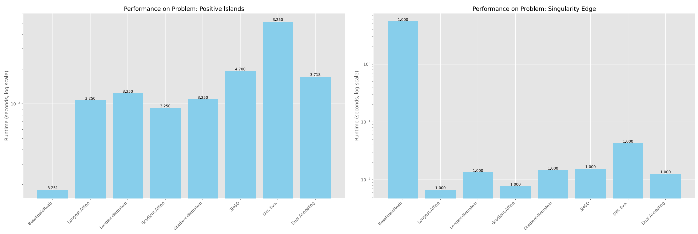


Figure 2: Plot of runtime performance for Pole Avoidance. Our method is significantly faster than the baseline (normalized runtime is very low, around 0.01-0.02x the baseline time) and numeric optimizations and finds a slightly better (lower) bound.

On Positive Islands in 3: Positive Islands The objective function is a simple polynomial $x^2 + 1$, but the feasible region consists of two small, disconnected spheres in the middle of a very large search box. Our algorithm likely struggles here because it has to spend the vast majority of its time subdividing a large, empty, infeasible space just to find the two tiny islands where solutions exist. The dReal baseline method might be more efficient at handling these kinds of disjoint feasible regions, which would explain why our method is comparatively slower on this specific problem.



(a) Positive islands runtime performance.

(b) Singularity edge runtime performance

Figure 3: (a) On certain problems, dReal defeats all other methods with marginal error. Our method is much slower than the dReal baseline (normalized runtime is around 6-7x slower the baseline time) while finding a similar bound. This is also the case for numeric optimizers. SHGO and Dual Annealing get the wrong answer. (b) On other problems, our method is able to match standard numerical optimization performance.

4.1.1. Aggregate Runtime Performance

Overall, our method

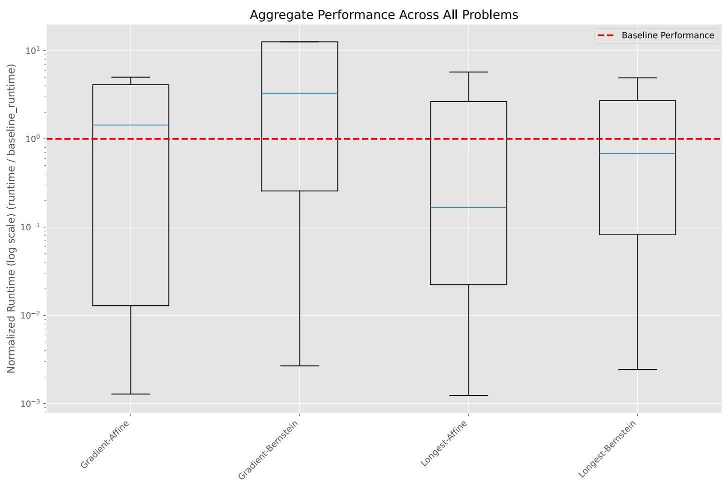


Figure 4: Plot of aggregate runtime performance across testing suite.

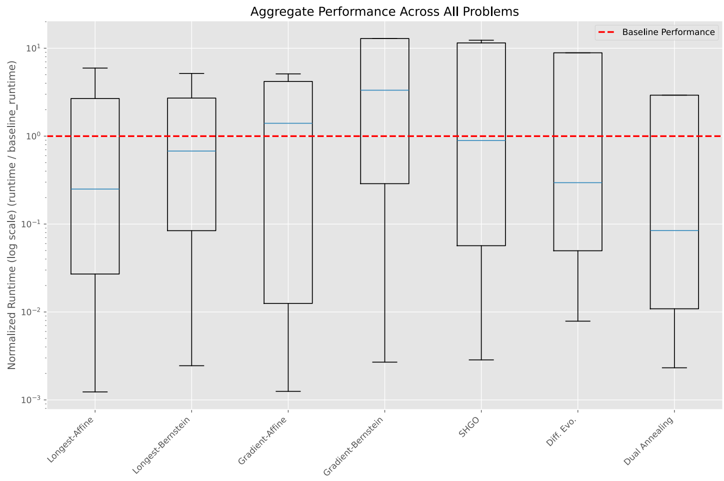


Figure 5: Plot of aggregate runtime performance across testing suite with standard numeric optimization methods.

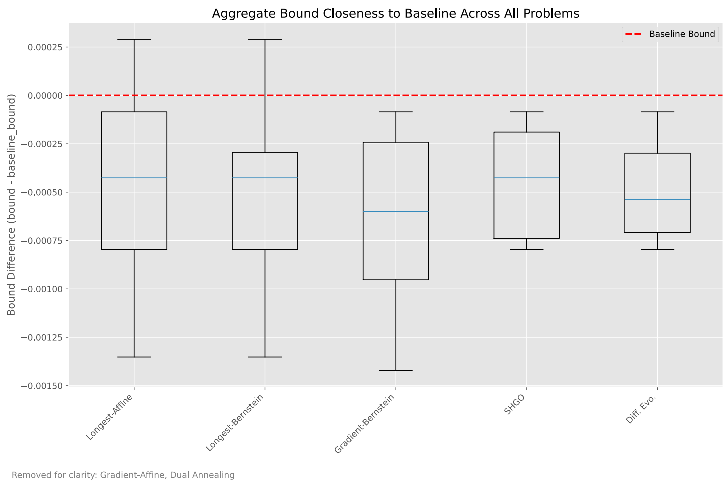


Figure 6: Plot of aggregate bound difference across testing suite. Note that the gradient-split and affine-bound heursitic combination has been ommitted as it tended to produce much larger (and incorrect) bounds on many problems. Dual annealing has been ommitted for a similar reason.

4.2. Design & Implementation Challenges

We discovered that the affine bounding heuristic can perform rapid minimization, but the heuristic alone is insufficient to uncover the actual minimum. We consider the following problem.

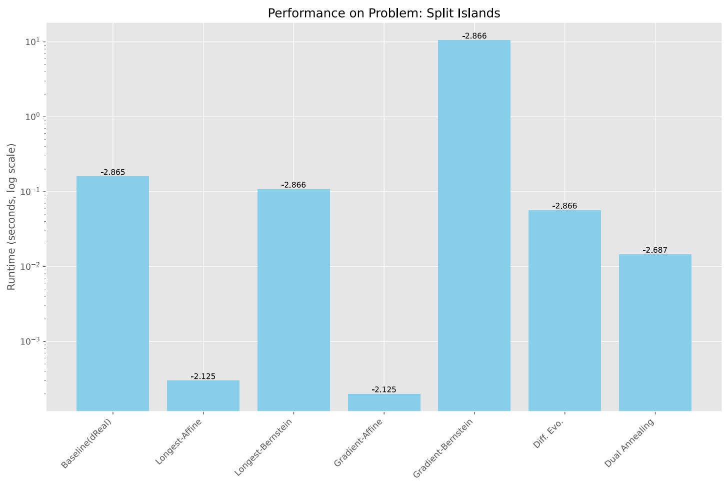
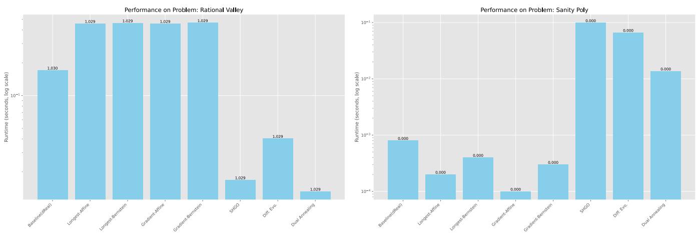


Figure 7: Plot of runtime performance for Split Islands. The algorithms with the affine interval bounding heuristic terminate quickly, but with the incorrect lower bound.

For some problems, numerical methods outperform solver-aided ones. The following plots demonstrate a coupling between our method and dReal with respect to runtime performance on the Rational Valley and Sanity Poly problems.



(a) Rational valley runtime performance.

(b) Sanity poly runtime performance

Figure 8: (a) On certain problems, solvers have a difficult time. (b) On others, the numerical methods suffer. All optimizers agree on the lower bound for both problems.

4.2.1. Selecting Bencmarks

We collected sample benchmarks to form our test suite from Wikipedia.[2] We currently have tests for: Sanity Poly, Sanity Rational, Rational Bowl, Rational Valley, Split Islands, Positive Islands, Singularity Edge, Pole Avoidance, Sparse Intersection, and also Himmelblau Ratio (but only for 3D).

4.3. Improvement Areas

We have several areas in which to improve our branch-and-bound method.

4.3.1. Adaptive Heuristics

Our results show that some heuristic combinations are fast but coarse, while others are slow but precise. We could define transition conditions from one splitting and bounding combo to another.[3] For example, an adaptive solver could begin with a prospecting phase using a fast heuristic like affine bounds with longest side bisect splitting. This would rapidly discard large regions of the search space and establish an initial upper bound on the global minimum. Once the rate of improvement slows down or the total volume of the active search boxes falls below a threshold, the algorithm could switch to a refining phase, using a more expensive but precise heuristic like the Bernstein bounds and gradient split to zero in on the true minimum.

4.3.2. Advanced Box Splitting Strategies

The current project uses bisecting on the longest box side which is ignorant to function behaviour, and splitting in the direction of greatest change in the objective function, gradient split. There's a rich area of research here that could yield significant performance gains. Additionally, it would be beneficial to compare our attempts at smart methods to merely random splitting. Another intelligent heuristic for nonlinear functions could be to split where the function has the greatest measure,that is where it is changing the most. We could implement a maximal smear splitter. For each dimension of a box, it would calculate the width of the objective function's range (its "smear"), an interval.[4] It would then split the box along the dimension that produces the widest range, as this is often where the most progress can be made in tight-

[2] https://en.wikipedia.org/wiki/Test_functions_for_optimization

[3] These conditions would be another heuristic themselves. We might wish to avoid second order heuristics.

[4] So, we could apply interval arithmetic SMT tools here.

ening the bounds. This could be particularly effective on problems like Rational Valley, where the function's behavior is complex and not well-aligned with the box's geometry.

4.3.3. Ordering Hybrid SMT and Numerical Methods

We treat the SMT and numerical methods as separate categories for comparison, but their strengths are complementary. Our hybrid approach of refining the search space early with numerical methods before making calls to dReal aims to be more powerful than either technique alone. However, dReal does have similar baked-in optimizations itself, and is not purely an SMT tool. Our current approach places the SMT-based dReal at the end of the pipeline, yet we could reverse this by using the SMT-based branch-and-bound algorithm to find and isolate guaranteed feasible regions before the final minimum. The SMT solver is excellent at handling complex constraints, as seen in the Positive Islands problem. Once it identifies a small box that is provably feasible, it could hand that box off to a fast numerical optimizer like SHGO or Differential Evolution to quickly find the local minimum within that feasible region.

5. Teamwork

Jaedon: Researched and tested different bounding algorithms across various sample problems. Explored Bernstein polynomials and full and partial affine arithmetic for pruning effectiveness, and considered different heuristics for box splitting: splitting along the objective function gradient, splitting along the gradient of the constraints, splitting along the widest dimension, splitting along an aggregate of these heuristics.

Ruizhe: Implemented the basic branch-and-bound algorithmic framework based on dReal, and conducted comparisons against dReal's Minimize procedure. Integrated the Bernstein method, the affine method and the gradient-guided splitting heuristic into the branch-and-bound framework. Initially designed simple test cases to demonstrate the advantages of our proposed methods.

Evan: Tested our implementations on various parameters and optimization functions to find strengths and weaknesses of our code. Produced visualizations in Blender (supported by scripts) by extracting boxes and current best points to provide better intuition to support the results from the tests.

lipson: Took meeting minutes for logistics: ensuring we had actionable tasks, planning the next meeting date, summarizing our current state and what we need to implement next. Unified python scripts with types and interfaces to be on the same page and communicate about our code. In the future, it could have been better to start with everyone on the same page about code quality practices, e.g. strong typing. Kept project documentation up to date: how to use our tool, install dReal container.

6. Course Topics

Emergent discovery of poor solver performances guided our optimization choices. While we learned some of the essential algorithms and techniques that have built up SMT tools, in practice, SMT tools can still feel like black boxes. As mentioned in class, solvers are tightly coupled tools that exchange modularity for performance. A lack of modularity impairs understanding.[5] Without a good understanding of dReal itself, we were optimizing slow SMT run times by reducing complexity of solver input, cutting down our search space using branch-and-bound techniques. With a more explicit understanding of the dReal solver backend, we can both guide the construction of our heuristic simplification techniques, and, even better, directly focus on the solver's shortcomings instead of pre-processing our input. That being said, we did encounter DPLL(ICP) as a case of the DPLL algorithm specialized to interval constraint propagation of SMT using interval arithmetic. Certainly, it is

[5]We experienced this on the project level, with untyped code.

hard to fit everything in during a quarter. We can either focus more theoretically on how a solver backend works versus more practically on how to build solver aided tools. The project provides an experiential way of learning the latter skill, however, as discussed above, we didn't quite have the knowledge or toolkit to unpack the SMT solver and build our optimization with respect to its internal implementation.

References

[1] S. Gao, S. Kong, E.M. Clarke, dReal: An SMT Solver for Nonlinear Theories over the Reals, in: Proceedings of the 24th International Conference on Automated Deduction (CADE), Springer, 2013: pp. 208–214. https://doi.org/10.1007/978-3-642-38574-2_14.