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.

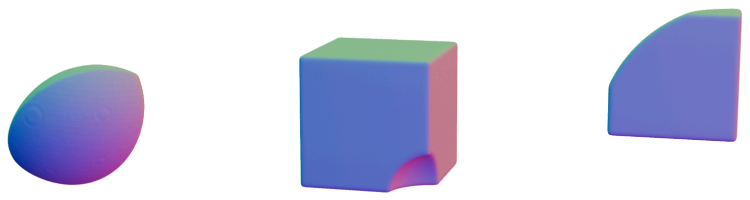

Figure 1: From left to right: hand-written example, Singular Edge, and Rational Bowl optimization search space renders.

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.

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.

Bernstein Polynomial 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.

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.

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.