dReal: An SMT Solver for Nonlinear Theories over the Reals*

Sicun Gao, Soonho Kong, and Edmund M. Clarke

Carnegie Mellon University, Pittsburgh, PA 15213

Abstract. We describe the open-source tool dReal, an SMT solver for nonlinear formulas over the reals. The tool can handle various nonlinear real functions such as polynomials, trigonometric functions, exponential functions, etc. dReal implements the framework of δ-complete decision procedures: It returns either unsat or δ-sat on input formulas, where δ is a numerical error bound specified by the user. dReal also produces certificates of correctness for both δ-sat (a solution) and unsat answers (a proof of unsatisfiability).

1 Introduction

SMT formulas over the real numbers can encode a wide range of problems in theorem proving and formal verification. Such formulas are very hard to solve when nonlinear functions are involved. Our recent work on δ-complete decision procedures provided a new framework for this problem [10,11]. We say a decision procedure is *δ-complete* for a set S of SMT formulas, where δ is a positive rational number, if for any φ from S, the procedure returns one of the following:

- unsat: φ is unsatisfiable.
- δ-sat: φ^δ is satisfiable.

Here, φ^δ is a syntactic variant of φ that encodes a notion of numerical perturbation on logic formulas [10]. With such relaxation, δ-complete decision procedures can fully exploit the power of scalable numerical algorithms to solve nonlinear problems, and at the same time provide suitable correctness guarantees for many correctness-critical problems. dReal implements this framework. It solves SMT problems over the reals with nonlinear functions, such as polynomials, sine, exponentiation, logarithm, etc. The tool is open-source[1], built on opensmt [5] for the high-level DPLL(T) framework, and realpaver [14] for the Interval Constraint Propagation algorithm. It returns unsat or δ-sat on input formulas, and the user can obtain certificates (proof of unsatisfiability or solution) for the answers.

In this paper we describe the usage, design, and some results of the tool.

Related Work. SMT solving for nonlinear formulas over the reals has gained much attention in recent years and many tools are now available. The symbolic approaches include Cylindrical Decomposition [6], with significant recent improvement [19,16], and Gröbner bases [20]. A drawback of symbolic algorithms is that it is restricted to arithmetic, namely polynomial constraints, with the exception of [1]. On the other hand, many practical solvers incorporate scalable numerical computations. Examples of numerical algorithms that have been exploited include optimization algorithms [4,18], interval-based algorithms [8,7,12], Bernstein polynomials [17], and linearization

* This research was sponsored by the National Science Foundation grants no. DMS1068829, no. CNS0926181 and no. CNS0931985, the GSRC under contract no. 1041377, the Semiconductor Research Corporation under contract no. 2005TJ1366, and the Office of Naval Research under award no. N000141010188.

[1] dReal is available at `http://dreal.cs.cmu.edu`.

algorithms [9]. All solvers show promising results on various nonlinear benchmarks. Our goal is to provide an open-source platform for the rigorous combination of numerical and symbolic algorithms under the framework of δ-complete decision procedures [10].

2 Usage

2.1 Input Format

We accept formulas in the standard SMT-LIB 2.0 format [2] with extensions. In addition to nonlinear arithmetic (polynomials), we allow transcendental functions such as sin, tan, arcsin, arctan, exp, log, pow, sinh. More nonlinear functions (for instance, solution of differential equations) can be added when needed, by providing the corresponding numerical evaluation algorithms. Floating-point numbers are allowed as constants in the formula.

Bound information on variables can be declared using a list of simple atomic formulas. For instance "`(assert (< 0 x))`", which sets $x \in (0, +\infty)$ at parsing time. Also, the user can set the precision by writing "`(set-info :precision 0.0001)`." The default precision is 10^{-3}, and can be set through command line.

Example 2.1. The following is an example input file. It is taken from the Flyspeck project [15]. (Filename `flyspeck/172.smt2`. Flyspeck `ID (6096597438b)`)

```
(set-logic QF_NRA)
(set-info :precision 0.001)
(declare-fun x () Real)
(assert (<= 3.0 x))
(assert (<= x 64.0))
(assert (not (> (- (* 2.0 3.14159265) (* 2.0 (* x (arcsin (* (cos
0.797) (sin (/ 3.14159265 x))))))) (+ (- 0.591 (* 0.0331 x))
(+ (* 0.506 (/ (- 1.26 1.0) (- 1.26 1.0))) 1.0)))))
(check-sat)
(exit)
```

2.2 Command Line Options

After building, `dReal` can be simply used through:

```
dReal [--verbose] [--proof] [--precision <double>] <filename>
```

The default output is unsat or delta-sat. When the flags are enabled, the following output will be provided.

- If `--verbose` is set, then the solver will output the detailed decision traces along with the solving process.
- If `--proof` is set, the solver produces an addition file "`filename.proof`" upon termination, and provides the following information.
  - If the answer is delta-sat, then `filename.proof` contains a witnessing solution, plugged into a δ-perturbation of the original formula, such that the correctness can be easily checked externally.

  - If the answer is `unsat`, then `filename.proof` contains a trace of the solving steps, which can be verified as a proof tree that establishes the unsatisfiability of the formula.
- The `--precision` flag gives the option of overwriting the default precision, and the one set in the benchmark.

When the `--proof` flag is set, the solver produces a file that certifies the answer. In the `delta-sat` case, the solution is plugged in the formula, and its correctness can be checked externally. For the `unsat` cases, we provide a proof checker that verifies the proof. It can be used with the following command:

```
proofcheck [--timeout <int>] <filename>
```

The proof checker will create a new folder called `filename.extra`, which contains auxiliary files needed. It is possible for the proof checking procedure to produce a large number of new files, so setting a timeout is important. By default, the timeout is `30min`. The proof checker will return either "proof verified" or "timeout".

Example 2.2. With default parameters, dReal solves the formula in Example 2.1 in `10ms`, returning unsat, on a machine with a 32-core 2.3GHz AMD Opteron Processor and 94GB of RAM. We then run proofcheck on the same machine. The proof checker returns "proof verified" in 10.08s, after making 8 branching steps and checking 77 axioms.

3 Design

3.1 The δ-Decision Problem

The standard decision problem is undecidable for SMT formulas over the reals with trigonometric functions. Instead, we proposed to focus on the so-called δ-decision problem, which relaxes the standard decision problem. Let δ be any positive rational number. On a given SMT formula φ, we ask for one of the following answers:

- unsat: φ is unsatisfiable.
- δ-sat: φ^δ is satisfiable.

When the two cases overlap, either answer can be returned. Here, φ^δ is called the δ-perturbation (or δ-weakening) of φ, which is formally defined as follows.

Definition 3.1 (δ-Weakening [10]). *Let $\delta \in \mathbb{Q}^+ \cup \{0\}$ be a constant and φ be a Σ_1-sentence in a standard form $\varphi := \exists^{\boldsymbol{I}}\boldsymbol{x}\ (\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k_i} f_{ij}(\boldsymbol{x}) = 0))$. The δ-weakening of φ defined as: $\varphi^\delta := \exists^{\boldsymbol{I}}\boldsymbol{x}\ (\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{k} |f_{ij}(\boldsymbol{x})| \leq \delta))$.*

Solving the δ-decision problem is as useful as the standard one for many problems. For instance, suppose we perform bounded model checking on hybrid systems, and encode safety properties as an SMT formula φ. Then following standard model checking techniques, if we decide that φ is unsat, then the system is indeed "safe" with in some bounds; if we decide that φ is δ-sat, then the system would become "unsafe" under some δ-perturbation on the system. In this way, when δ is reasonably small, we have essentially taken into account the robustness properties of the system, and can justifiably conclude that the system is unsafe in practice.

3.2 DPLL⟨ICP⟩

Interval Constraint Propagation (ICP) [3] is a constraint solving algorithm that finds solutions of real constraints using a "branch-and-prune" method, combining interval arithmetic and constraint propagation. The idea is to use interval extensions of functions to "prune" out sets of points that are not in the solution set, and "branch" on intervals when such pruning can not be done, until a small enough box that may contain a solution is found. In a DPLL(T) framework, ICP can be used as the theory solver that checks the consistency of a set of theory atoms. We use opensmt [5] for the general DPLL(T) framework, and integrate realpaver [14] which performs ICP. We now describe the design of the interface. A high-level structure of the theory solver is shown in Algorithm 1.

Algorithm 1: Theory Solving in DPLL⟨ICP⟩

input : A conjunction of theory atoms, seen as constraints, $c_1(x_1, ..., x_n), ..., c_m(x_1, ..., x_n)$, the initial interval bounds on all variables $B^0 = I_1^0 \times \cdots \times I_n^0$, box stack $S = \emptyset$, and precision $\delta \in \mathbb{Q}^+$.
output: δ-sat, or unsat with learned conflict clauses.

S.push(B_0);
while $S \neq \emptyset$ **do**
    $B \leftarrow S$.pop() ;
    while $\exists 1 \leq i \leq m, B \neq \text{Prune}(B, c_i)$ **do**
        `//Pruning without branching, used as the assert() function.` $B \leftarrow \text{Prune}(B, c_i)$;
    end
    `//The` ε `below is computed from` δ `and the Lipschitz constants of functions beforehand.`
    if $B \neq \emptyset$ **then**
        if $\exists 1 \leq i \leq n, |I_i| \geq \varepsilon$ **then**
            $\{B_1, B_2\} \leftarrow \text{Branch}(B, i)$; `//Splitting on the intervals` S.push($\{B_1, B_2\}$);
        else
            return δ-sat; `//Complete check() is successful.`
        end
    end
end
return unsat;

Check and Assert. For incomplete checks in the assert function, we use the pruning operator provided in ICP to contract the interval assignments on all the variables, by eliminating the boxes in the domain that do not contain any solutions. At complete checks, we perform both pruning and branching, and look for one interval solution of the system. That is, we prune and branch on the interval assignment of all variables, and stop when either we have obtained an interval vector that is smaller than the preset error bound, or when we have traversed all the possible branching on the interval assignments.

Backtracking and Learning. We maintain a stack of assignments on the variables, which are mappings from variables to unions of intervals. When we reach a conflict, we backtrack to the previous environment in the pushed stack. We also collect all the constraints that have appeared in the pruning process leading to the conflict. We then turn this subset of constraints into a learned clause and add it to the original formula.

Witness for δ-Satisfiabillity. When the answer is δ-sat on $\varphi(\boldsymbol{x})$, we provide a solution $\boldsymbol{a} \in \mathbb{R}^n$, such that $\varphi^\delta(\boldsymbol{a})$ is a ground formula that can be easily checked to be true. It is important to note that the solution witnesses δ-satisfiability, instead of standard satisfiability of the original formula. While the latter problem is undecidable, *any* point in the interval assignment returned by ICP can witness the satisfiability of φ^δ when the intervals are smaller than an appropriate error bound.

Proofs of Unsatisfiability. When the answer is unsat, we produce a proof tree that can be verified to establish the validity of the negation of the formula, i.e., $\forall \boldsymbol{x} \neg \varphi(\boldsymbol{x})$. We devised a simple first-order natural deduction system, and transform the computation trace of the solving process into a proof tree. We then use interval arithmetic and simple rules to check the correctness of the proof tree. The proof check procedure recursively divide the problem into subproblems with smaller domains. More details can be found in [13].

4 Results

Problem#	#OP	Time_S	Result	Trace Size	PC	#PA	#SP	Time_PC	#D
506	49	0:00.01	UNSAT	519	Y	3,108	3,107	190.200	9
504	48	0:00.01	UNSAT	507	Y	2,322	2,321	172.250	9
746	2,729	0:00.22	UNSAT	20,402	Y	134	135	156.940	99
785	81	0:00.79	UNSAT	2,530,262	Y	1,968	1,454	100.620	5
505	48	0:00.01	UNSAT	477	Y	1,390	1,389	84.030	9
814	96	0:00.50	UNSAT	1,349,482	Y	885	638	79.010	5
783	832	0:00.06	UNSAT	6,386	Y	211	210	57.890	9
815	96	0:00.48	UNSAT	1,394,542	Y	912	688	45.620	5
760	2,792	0:00.22	UNSAT	20,991	Y	71	70	34.470	9
816	97	0:00.15	UNSAT	423,074	Y	335	254	30.310	5
260	90	0:45.10	UNSAT	306,508,373	N	—	—	—	–
884	94	0:25.75	UNSAT	181,766,839	N	—	—	—	–
461	36	0:25.20	UNSAT	133,865,608	N	—	—	—	–
871	80,230	0:16.38	UNSAT	610,809	N	—	—	—	–
525	43	4:38.01	δ-SAT	—	–	—	—	—	–

Table 1: Experimental results. #OP = Number of nonlinear operators in the problem, TIME_S = Solving time in seconds, TO = Timeout (30min), PC = Proof Checked, #PA = Number of proved axioms, #SP = Number of subproblems generated by proof checking, TIME_PC = Proof-checking time in seconds, #D = Number of iteration depth required in proof checking.

Besides solving the standard benchmarks [16] (data shown on the tool website), we managed to solve many challenging nonlinear benchmarks from the Flyspeck project [15] for the formal proof of the Kepler conjecture. The following is a typical formula:

$$\forall \boldsymbol{x} \in [2, 2.51]^6. \Big(-\frac{\pi - 4\arctan\frac{\sqrt{2}}{5}}{12\sqrt{2}}\sqrt{\Delta(\boldsymbol{x})} + \frac{2}{3}\sum_{i=0}^{3}\arctan\frac{\sqrt{\Delta(\boldsymbol{x})}}{a_i(\boldsymbol{x})} \le -\frac{\pi}{3} + 4\arctan\frac{\sqrt{2}}{5}\Big)$$

where $a_i(\boldsymbol{x})$ are quadratic and $\Delta(\boldsymbol{x})$ is the determinant of a nonlinear matrix. We solved 828 out of the 916 formulas (returning unsat) with a timeout of 5 minutes and $\delta = 10^{-3}$, without domainspecific heuristics. The proof traces of these formulas can be large. In Table 1, we list some of the representative benchmarks to show scalability. Complete tables are on the tool page.

References

1. B. Akbarpour and L. C. Paulson. Metitarski: An automatic prover for the elementary functions. In *AISC/MKM/Calculemus*, pages 217–231, 2008.
2. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.
3. F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 16. Elsevier, 2006.
4. C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell, and A. Rubio. Solving non-linear polynomial arithmetic via sat modulo linear arithmetic. In *CADE*, pages 294–305, 2009.
5. R. Bruttomesso, E. Pek, N. Sharygina, and A. Tsitovich. The opensmt solver. In J. Esparza and R. Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2010.
6. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183, 1975.
7. A. Eggers, M. Fränzle, and C. Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *ATVA*, pages 171–185, 2008.
8. M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.
9. M. K. Ganai and F. Ivančić. Efficient decision procedure for non-linear arithmetic constraints using cordic. In *Formal Methods in Computer Aided Design (FMCAD)*, 2009.
10. S. Gao, J. Avigad, and E. M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In *IJCAR*, pages 286–300, 2012.
11. S. Gao, J. Avigad, and E. M. Clarke. Delta-decidability over the reals. In *LICS*, pages 305–314, 2012.
12. S. Gao, M. Ganai, F. Ivancic, A. Gupta, S. Sankaranarayanan, and E. Clarke. Integrating ICP and LRA solvers for deciding nonlinear real arithmetic. In *FMCAD*, 2010.
13. S. Gao, S. Kong, M. Wang, and E. Clarke. Extracting proofs from a numerically-driven decision procedure, 2013. CMU SCS Technical Report CMU-CS-13-104.
14. L. Granvilliers and F. Benhamou. Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques. *ACM Trans. Math. Softw.*, 32(1):138–156, 2006.
15. T. C. Hales. Introduction to the flyspeck project. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, volume 05021 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.
16. D. Jovanovic and L. M. de Moura. Solving non-linear arithmetic. In *IJCAR*, pages 339–354, 2012.
17. C. Muñoz and A. Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 2012. Accepted for publication.
18. P. Nuzzo, A. Puggelli, S. A. Seshia, and A. L. Sangiovanni-Vincentelli. Calcs: Smt solving for non-linear convex constraints. In R. Bloem and N. Sharygina, editors, *FMCAD*, pages 71–79. IEEE, 2010.
19. G. O. Passmore and P. B. Jackson. Combined decision techniques for the existential theory of the reals. In *Calculemus/MKM*, pages 122–137, 2009.
20. A. Platzer, J.-D. Quesel, and P. Rümmer. Real world verification. In *CADE*, pages 485–501, 2009.