CRTSolver: Solving Non-Linear Integer Equations using the Chinese Remainder Theorem

An Extended Abstract

Martin Brain[1], Maheen Matin[1]

[1]City St. George's, University of London, Northampton Square, London, EC1V 0HB, United Kingdom

Abstract

The SMT-LIB theory of integers allows for non-linear terms and is thus undecideable in the general case. This is a practical problem but also discourages work on the decideable and easy fragments of the theory. In this extended abstract we present a simple semi-decision procedure for sets of polynomial integer equations that shows significant promise in preliminary experiments with synthetic benchmarks. We hope that this will illuminate the considerable improvements that are possible with well known techniques from the Diophantine equation and number theory literature.

Keywords

Satisfiability modulo theories, Non-linear integer equations, Diophantine equations

1. Introduction

A reasonable performance floor for SMT solvers is that they should be faster than a human solving the same problem with a pen and paper. In the case of non-linear expressions over theory of integers this floor is not always met. For example consider the following problem:

$$2x^3 + 3x^2 + 6x + 8 = 0$$

In our experiments we found that cvc5 timed out after 30 seconds. Observing that all coefficients are positive including the constant so x must be strictly negative and then trying decreasing values ($x = -1, x = -2, \dots$) gives an answer very quickly.

Hilbert's tenth problem asked (in modern terminology) whether there is an decision procedure for Diophantine equations (a polynomial equation possessing only integer coefficients). The MRDP theorem gives a construction from recursively enumerable sets to Diophantine equations, thus showing the undecideability of satisfiability.

This has an immediate practical consequence of their being no algorithms for the general case but we also believe that it has had chilling effect on the theory in general. We believe there are many interesting and useful fragments of the theory of integers for which there are practical decision procedures, semi-decision procedures or good heuristics. In this paper we will consider one such fragment, existential polynomial equations. In Section 2 we propose an abstraction-refinement style algorithm that uses the Chinese Remainder Theorem. Section 3 describes some preliminary experiments using our implementation of the algorithm. Finally Section 4 discusses the future potential of this approach.

2. Algorithm

This section gives a semi-decision procedure for polynomial integer equations. It is based on the following two elementary observations:

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK

martin.brain@citystgeorges.ac.uk (M. Brain); maheen.matin@city.ac.uk (M. Matin)

0000-0003-4216-7151 (M. Brain); 0009-0005-8263-7041 (M. Matin)

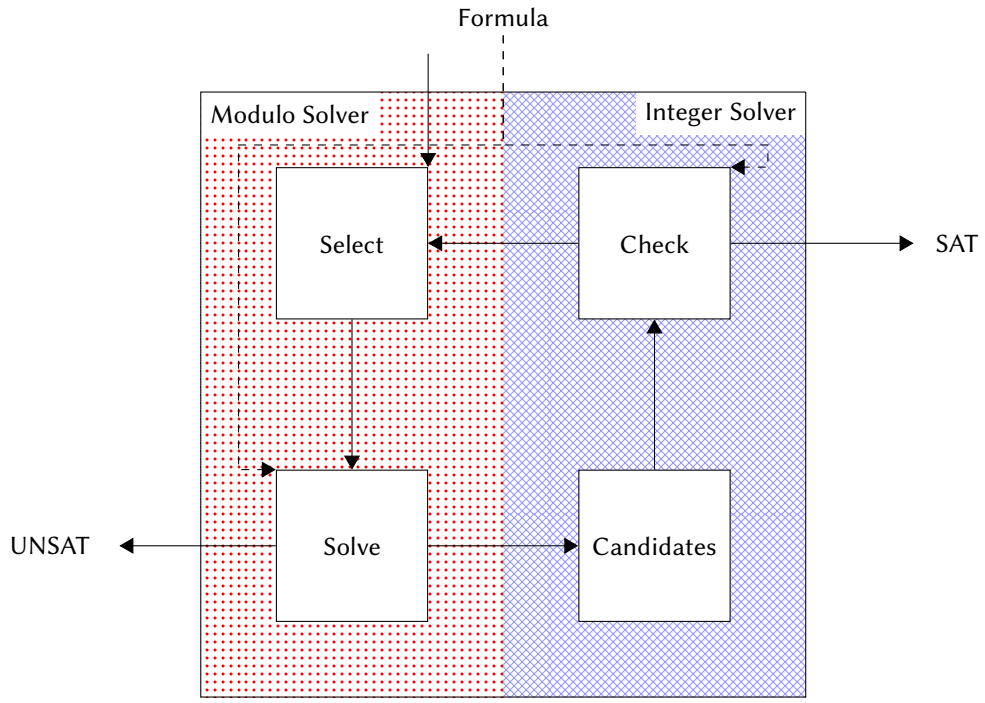

Figure 1: A flow-chart description of a Abstraction/Refinement style semi-decision procedure for Diophantine equations

1. If a set of equations is satisfiable over the integers then it will be satisfiable modulo any number. We use the contraposition – if a set of equations is unsatisfiable over a specific modulus then it must be unsatifiable over the integers.
2. If a set of equations is solvable modulo p and q with p and q being coprime then they will be solvable modulo $p * q$. This is a consequence of the Chinese Remainder Theorem.

These are used to create an abstraction-refinement style loop using a pair of solvers. One computes the solutions modulo a product of primes (giving an over-approximation of the true satisfying assignments) and the other checks these candidates. The algorithm consists of a loop of four phases, illustrated in Figure 1:

Select A new modulo m_i, co-prime with all previous modulos is chosen. We currently use an ascending list of primes, starting from $m_0 = 2$.

Solve A copy of the set of equations computed $mod\ m_i$ is added to the modulo solver. Then the modulo solver is checked for satisfiability. If it is unsatisfiable then the equations are unsolvable modulo in at least one of m_i's and thus the original problem is unsatisfiable. If it is satisfiable then the model will give solutions for the equations modulo every m_i used so far.

Candidates The Chinese Remainder Theorem is used to compute a solution modulo Πm_i. These are used to build a number of candidates over $\mathbb{Z}$, each of which is the solution modulo Πm_i plus $k\Pi m_i$. Currently we produce candidates for $k \in [-2, 2]$.

Check The original equations are evaluated using the candidate solutions. If this gives a satisfying assignment then the original system is satisfiable. If all of the candidates fail to satisfy the original equations then a new modulus must be used.

The algorithm above is a semi-decision procedure – if it terminates then it will give the correct result. However the termination of the algorithm is less obvious than it might appear. There are equations

MB: If have a worked example and want to include it, here would be good.

which are satisfiable modulo any m but are not satisfiable over $\mathbb{Z}$. For example:

$$w^2 + x^2 + y^2 + z^2 + 1 = 0$$

clearly as no solutions over $\mathbb{Z}$ but by Lagrange's Four Square Theorem there are solutions for any modulus m.

3. Results

We have implemented the algorithm in Section 2 in a tool called `CRTSolver` which is available at https://github.com/maheenmatin/CRTSolver under the MIT license. It uses two instances of cvc5, one to solve the modulo sub-problems and a second to check the candidate results. The following options are available: Integer Mode and Bit-Vector Mode.

Integer Mode attempts to solve the modulo subproblem using the theory of Quantifier-Free Nonlinear Integer Arithmetic, where the variables are of integer sort and integer operations are used. Bit-Vector Mode uses the theory of Quantifier-Free Bit-Vector Logic, where the variables are of bit-vector sort (with a fixed bit-width calculated as a function of the current prime) and bit-vector operations are used. For the checking of candidate results, the theory of Quantifier-Free Nonlinear Integer Arithmetic is used for both Integer Mode and Bit-Vector Mode.

We have created a set of benchmarks that evaluate the performance of CRTSolver in both available modes on non-linear integer equations, in comparison with two widely used and state-of-the-art SMT solvers, Z3 and cvc5. The benchmarks are time and success rate, where time simply refers to the total runtime for a given equation in milliseconds. We define a successful solving of a given equation as termination with `SAT` or `UNSAT` - if a solver terminates with `UNKNOWN` due to time constraints, memory constraints, or internal error, then we define this as an unsuccessful solving.

In Table 1 we give a comparison with Z3 [1] and cvc5 [2]. The performance evaluation was conducted using a Juypter Notebook file on Visual Studio Code (version 1.100). WSL2 (Ubuntu) was used on a Windows 11 Home (version 10.0.26100) PC with an Intel i5-11400 processor and 16GB of RAM. The respective Python APIs for the latest versions of Z3 (version 4.14.1.0) and cvc5 (version 1.2.1) at the time of testing were used.

MB: Check these details

All solvers had their available memory limited to 4GB and were given the same time-out value (the time limit for each `check-sat`) of 10 seconds. They were all given the same 30 test files, which were in the form of SMT2 files containing non-linear integer equations. Files were divided into 3 groups: equations involving 1 variable, equations involving 2 variables and equations involving 3 variables. Each of these groups contained sub-groups of quadratic and cubic equations. There was a mixture of both satisfiable and unsatisfiable equations - however, there was a slight bias for the frequency of unsatisfiable cubic equations as these were found to be the most demanding equations, thus making them especially suitable for evaluating performance.

MM: Should we mention the integer overflow error to this extent?

For each equation, we give the number of variables present, the degree, and whether or not the equation is satisfiable. CRTSolver's Integer Mode and Bit-Vector mode have been represented using the column headings `CRT-INT` and `CRT-BV` respectively. For each solver, the time taken for the equation is given. If the solver was successful in solving the given equation, we simply provide this in seconds. Where the runtime is greater than or equal to 1 second it is represented with 3 decimal places, and if less than 1 second it is represented in scientific notation with 3 significant figures. However, the solver may be unsuccessful due to a number of reasons, therefore terminating with `UNKNOWN`. In the case of exceeding the time-out value, we denote this with `T/O` - note that in every such instance the runtime is equal to the time-out value of 10 seconds. Finally, in the case of integer overflow during solving, we denote this with `U` and provide the runtime before termination with integer overflow. These results are also shown in a cactus plot in Figure 2. All results were produced using v1.1.0 of CRTSolver.

MB: Check that the notation here matches the table

These results suggest that there using CRTSolver realistic potential in using CRTSolver in Bit-Vector Mode as a solver that offers equal or better performance than existing solvers for non-linear integer equations. However, the Integer Mode of CRTSolver can be safely concluded as having no real potential.

MM: Please check the following equations for satis-

Table 1
Performance Comparison between CRTSolver, Z3 and cvc5.

Variables	Degree	SAT	CRT-INT	CRT-BV	Z3	cvc5
1	2	UNSAT	T/O	1.71×10^{-02}	1.34×10^{-02}	3.48×10^{-02}
1	2	SAT	1.98×10^{-01}	1.82×10^{-02}	1.78×10^{-02}	9.67×10^{-03}
1	2	SAT	5.58×10^{-01}	4.11×10^{-03}	5.24×10^{-03}	1.91×10^{-02}
1	2	SAT	T/O	1.75×10^{-02}	3.89×10^{-03}	8.65×10^{-03}
1	2	UNSAT	T/O	3.00×10^{-03}	8.58×10^{-03}	1.77×10^{-02}
1	2	SAT	5.63×10^{-01}	6.05×10^{-03}	4.91×10^{-03}	6.77×10^{-03}
1	2	UNSAT	T/O	1.45×10^{-02}	3.64×10^{-03}	1.02×10^{-02}
1	2	UNSAT	T/O	1.86×10^{-02}	3.90×10^{-02}	2.10×10^{-02}
1	3	UNSAT	T/O	6.49×10^{-02}	4.56×10^{-03}	T/O
1	3	SAT	6.68×10^{-03}	1.68×10^{-02}	8.48×10^{-03}	7.61×10^{-03}
1	3	SAT	8.83×10^{-03}	6.39×10^{-03}	6.99×10^{-03}	8.80×10^{-03}
1	3	SAT	4.10×10^{-01}	1.21×10^{-02}	4.59×10^{-03}	4.04×10^{-01}
1	3	UNSAT	T/O	9.24×10^{-02}	1.86×10^{-02}	T/O
1	3	UNSAT	T/O	2.19×10^{-02}	1.80×10^{-02}	T/O
1	3	UNSAT	T/O	3.14×10^{-03}	2.36×10^{-03}	6.34×10^{-03}
2	2	UNSAT	T/O	U (8.26×10^{-01})	8.27×10^{-03}	3.34×10^{-01}
2	2	SAT	T/O	U (5.65×10^{-01})	2.48×10^{-03}	2.68×10^{-02}
2	2	SAT	7.20×10^{-01}	1.11×10^{-01}	2.94×10^{-03}	6.97×10^{-02}
2	2	SAT	2.32×10^{-01}	2.56×10^{-02}	5.54×10^{-03}	1.820
2	2	UNSAT	T/O	U (5.64×10^{-01})	3.77×10^{-03}	5.18×10^{-01}
2	3	SAT	3.70×10^{-01}	8.81×10^{-02}	3.82×10^{-03}	4.33×10^{-01}
2	3	UNSAT	T/O	U (1.212)	T/O	T/O
2	3	UNSAT	T/O	U (9.07×10^{-01})	T/O	T/O
2	3	UNSAT	T/O	7.41×10^{-02}	5.22×10^{-03}	6.93×10^{-02}
2	3	SAT	1.47×10^{-02}	2.92×10^{-02}	1.96×10^{-03}	7.92×10^{-03}
2	3	UNSAT	T/O	8.22×10^{-02}	T/O	1.94×10^{-01}
2	3	UNSAT	4.97×10^{-03}	4.54×10^{-03}	2.04×10^{-03}	6.17×10^{-03}
3	2	SAT	2.86×10^{-01}	8.91×10^{-02}	4.36×10^{-03}	1.683
3	2	UNSAT	T/O	1.04×10^{-01}	3.41×10^{-03}	6.14×10^{-03}
3	2	SAT	T/O	3.66×10^{-01}	5.50×10^{-03}	T/O
3	2	SAT	2.55×10^{-01}	5.80×10^{-02}	3.35×10^{-03}	5.07×10^{-03}
3	3	UNKNOWN	T/O	U (3.196)	T/O	T/O
3	3	UNSAT	T/O	2.52×10^{-01}	3.75×10^{-03}	T/O
3	3	SAT	1.47×10^{-01}	1.73×10^{-01}	5.52×10^{-03}	8.13×10^{-02}
3	3	SAT	8.47×10^{-02}	7.20×10^{-02}	2.63×10^{-03}	1.04×10^{-02}
3	3	SAT	2.89×10^{-01}	1.76×10^{-01}	2.72×10^{-03}	1.580
3	3	UNKNOWN	T/O	U (2.651)	T/O	T/O
3	3	UNKNOWN	T/O	U (3.543)	T/O	T/O

4. Conclusion

MB: Write when results are available

It works (on the limited set of benchmarks we have tried). It shows enough promise to be investigated further. We want to try to implement it in an actual solver.

There are many open questions and possible future directions for this work:

- Using ascending primes as modulo is correct but neglects various potential optimisations. For example primes of the form $2^n \pm k$ for small k allow for efficient computation of bit-vector remainder. Also some moduli simplify various powers, for example $x^4 mod 20$ can only have one of four possible values.
- There are a range of approaches to encoding the modulo equations. If bit-vectors are used there are questions about what bit-widths are used and how often modulo reductions are performed. There are a range of techniques used to optimise the performance of cryptographic primitives

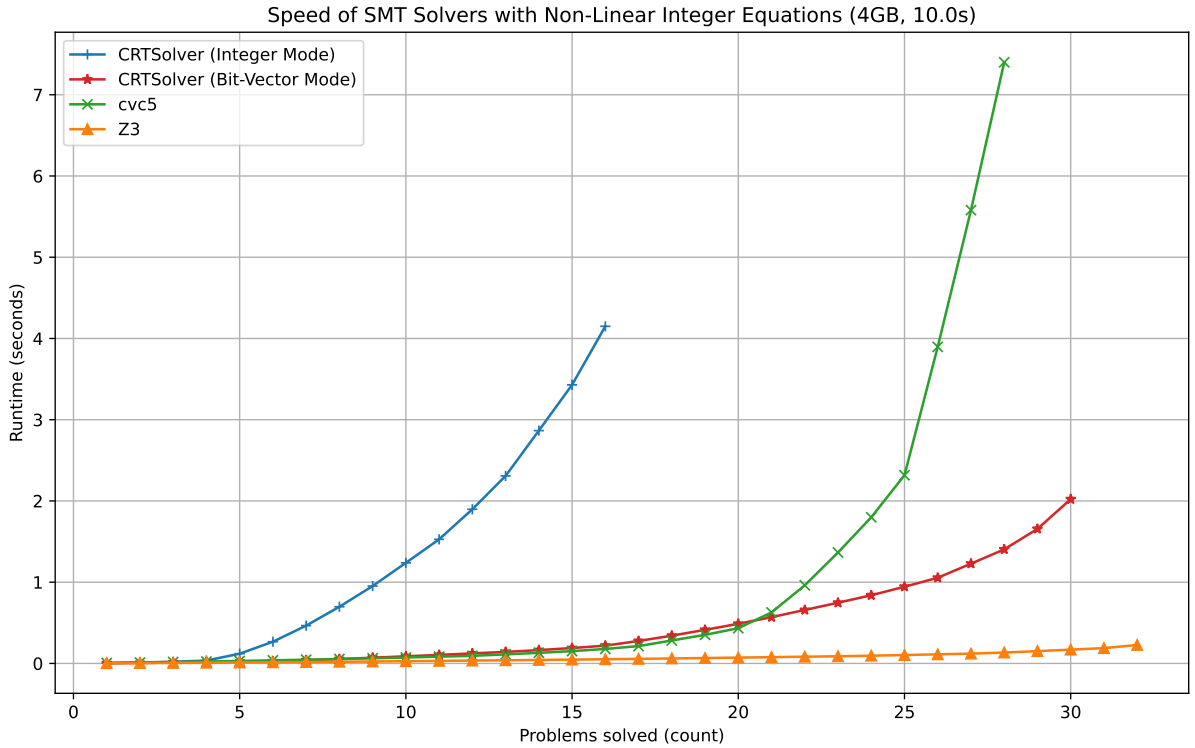

Figure 2: A cactus plot of the results.

that might be used. Alternatively the finite field solver of cvc5 might be an option.

- The selection of candidates can likely be improved through caching partially successful and unsuccessful candidates. Alternatively computation of candidates could be treated as a separate non-linear sub-problem, similarly to the handling of quantifiers in MBQI.
- The current algorithm takes minimal information from the failed candidates to inform the selection of a new modulus. This is an area that can clearly be strengthened to better fit with the abstraction-refinement style of the algorithm.
- Moving beyond equations, can this technique be extended to non-equalities ($\neq$), inequalities ($\leq$) and non-polynomial expressions?

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] L. de Moura, N. Bjørner, Z3: An efficient smt solver, in: C. R. Ramakrishnan, J. Rehof (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 337–340.

[2] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength SMT solver, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as

Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I, volume 13243 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 415–442. URL: https://doi.org/10.1007/978-3-030-99524-9_24. doi:10.1007/978-3-030-99524-9_24.