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

MM: We both have the same address, so please advise on the preferred format

Abstract

The theory of integers allows for non-linear terms and is thus undecideable in the general case. This is not only 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 the polynomial integer equations which 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.

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 but observing that small integers (i.e. 0, 1, -1) result in outputs greater than or less than zero - but never exactly zero, will allow many humans to solve it faster.

The undecidability of non-linear integer equations is a characteristic determined by Hilbert's tenth problem and the subsequent MRDP theorem (also known as Matiyasevich's theorem). Hilbert's tenth problem asks if there is a general formula that, for a given Diophantine equation (a polynomial equation possessing only integer coefficients), can determine the existence of an integer solution.

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.

MM: Please do check the intro and make/propose changes as necessary

MM: only cvc5 times out on this equation, not Z3. There are no equations for which Z3 times out and CRTSolver doesn't

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 semi-decision procedure for polynomial integer equations.

.

1. If a set of equations is satisfiable over the integers then it will be satisfiable modulo any number. We use the converse – 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 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

Pick Prime The next prime number p from the list of ascending primes is picked. On the first iteration, $p = 2$

Solve Modulo Subproblems The original equation is reduced modulo p by applying mod p to each term individually

Extract Candidates If satisfiable, candidate values for each variable in the subproblem are extracted from the solver

Check Candidates The Chinese Remainder Thoerem is used to compute several new candidates for each variable based on the previously extracted candidates. Each possible combination of candidate values is checked for satisfiability against the original problem

We should probably say something about correctness, complexity or completeness. That might be "We don't know".

I think there are theorems which says something like:

- This is a semi-decision procedure (if it terminates then it is correct)
- If all of the variables are bounded so they have a finite range then the algorithm will terminate

but I am unsure if we will have time to prove these.

MB: Create a flow-chart of the phases. I can do this if you don't feel like doing it

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 LICENSE-GOES-HERE. 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

MM: I will prioritise the other tasks. In the mean-time, I will add a hand-drawn flowchart to the repo

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

MB: Think about this.

Table 1
Performance Comparison between CRTSolver, Z3 and cvc5.

Equation	Variables	Degree	SAT	CRT-INT	CRT-BV	Z3	cvc5
$3x^2 + 7x + 8 = 0$	1	2	No	T/O	15.053	4.045	9.739
?	?	?	?	?	?	?	?

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 (de Moura and Bjørner, 2008) and cvc5 (Barbosa et al., 2022). 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.

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 10000 milliseconds. 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.

For each equation, we give the full mathematical representation, as well as 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 the time taken in milliseconds. 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 and in the case of exceeding memory constraints, we denote this with M/O. Finally, in the case of integer overflow during solving, we denote this with I/O. These results are also shown in a cactus plot in Figure 2.

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: Added Harvard in-text citation and @inproceedings in .bib

MM: Let me know you're satisfied with the table. If so, I'll fill in the rows

4. Conclusion

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:

- Are there better encodings for solving the modulo sub problems in the BV solver?
- We select primes sequentially, is that the best thing to do or are there better strategies given the encoding, the constants in the problem, how previous candidates have failed, etc.
- How many candidates should be checked, how should they be picked?
- Can the failed candidates be used to improve the modulo sub problems?
- Can this technique be extended to non-equalities ($\neq$), inequalities ($\leq$) and non-polynomial expressions?

MB: This is in note form, please try to write. This should not be more than a page. Half a page is probably about right.

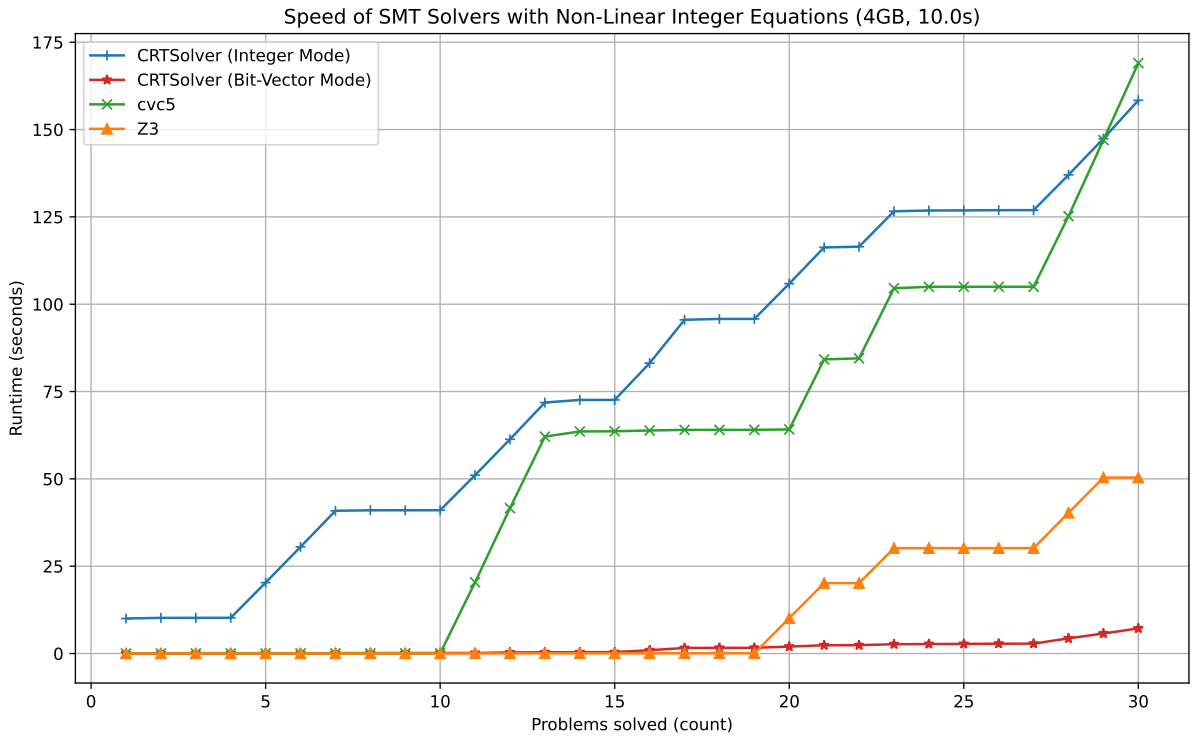

Figure 2: A cactus plot of the results.

Declaration on Generative AI

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

Or (by using the activity taxonomy in ceur-ws.org/genai-tax.html):
During the preparation of this work, the author(s) used X-GPT-4 and Gramby in order to: Grammar and spelling check. Further, the author(s) used X-AI-IMG for figures 3 and 4 in order to: Generate images. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

MB: We need to fill this out. I haven't used any generative AI tools.

References