Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Projeto Orientado em Computação II

Integrating an automata-based Presburger arithmetic decision procedure into the cvc5 theorem prover

Student: Luís Felipe Ramos Ferreira

Advisor: Haniel Barbosa

Belo Horizonte - Minas Gerais
2024

Summary

Resumo

Satisfatibilidade Módulo Teorias (SMT) é um campo em crescimento na ciência da computação e na lógica. Dado uma fórmula matemática, resolver o problema SMT significa determinar se a fórmula é satisfatível no contexto de uma teoria específica. Uma dessas teorias é a aritmética de Presburger (também conhecida como Aritmética Linear Inteira), que é a teoria de primeira ordem dos inteiros com adição. Um método para definir a satisfatibilidade de fórmulas nessa teoria envolve o uso de autômatos finitos [10]. Neste projeto, estendemos o solucionador cvc5 [2] incorporando uma ferramenta baseada em autômatos para resolver fórmulas de aritmética de Presburger, e comparamos com solucionadores do estado da arte como Lash [42] e Amaya [24]. Nossa ferramenta é baseada na biblioteca de autômatos Mata [15]. Como trabalho futuro, planejamos também adicionar as etapas de pré-processamento descritas em [24] para a construção de autômatos mais eficientes.

Palavras chave: Satisfabilidade Módulo Teorias, Aritmética de Presburger, Autômatos

Abstract

Satisfiability Modulo Theories (SMT) is a growing field in computer science and logic. Given a mathematical formula, solving the SMT problem means determining whether the formula is satisfiable within the context of a specific theory. One such theory is Presburger arithmetic (also known as Linear Integer Arithmetic), which is the first-order theory of integers with addition. A method for defining the satisfiability of formulae in this theory involves the use of finite automata [10]. In this project, we extend the cvc5 solver [2] by incorporating an automata-based tool for solving Presburger arithmetic formulae, and compare it with state of the art solvers such as Lash [42] and Amaya [24]. Our tool is based on the Mata automata library [15]. As future work, we plan to also add the preprocessing steps described in [24] for the construction of more efficient automata.

Keywords: Satisfiability Modulo Theories, Presburger arithmetic, Automata

Integrating an automata-based Presburger arithmetic decision procedure into the cvc5 theorem prover

Luís Felipe Ramos Ferreira[1], **Haniel Barbosa**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

{luisfeliperamos, hbarbosa}@dcc.ufmg.br

1. Introduction

Satisfiability Modulo Theories (SMT) is a growing field of study in computer science and mathematical logic [3]. In this research area, computer scientists and mathematicians aim to develop and create algorithms for automated reasoning over formulae defined over a set of theories. One of such theories is Presburger arithmetic (also know as Linear Integer Arithmetic, or LIA for short) [23, 35], which is the first-order theory of integers with addition.

The usage of LIA in SMT solvers has, besides its theorical importance, several practical applications in fields like databases [17], compiler optimizations [6] and resource allocation [25]. There are several different approaches to deal with such type of problems in the context of SMT, such as the quantifier elimination, Branch and Bound, etc [28].

There are also approaches for solving LIA problems based on automata theory [10, 12, 18]. The main idea of this approach is to construct a finite automata from the original input formula in a way such that the set of words accepted by the automata will correspond to the set of solutions for the original problem. This construction allows the usage of several automata theory concepts and algorithms.

As far as the authors are aware of, there are two open source implementations of such automata-based solvers available for usage, Lash [42] and Amaya [24]. Both of them will be evaluated and studied for a better understanding of how they work.

As the main part of this monograph, a self written solver using the Mata [15] automata library as a backend shall be implemented inside the cvc5 environment. This solver will be heavily inspired by the two aforementioned solvers and will be based on the algorithms and optimizations presented in [26], which allowed for the construction of efficient automata and was able to compete with other methods of solving LIA formulae in solvers like cvc5 [2] and Z3 [16].

1.1. Objectives

This monograph has the objetive of integrating into the cvc5 theorem prover a functional and performative Presburger arithmetic solver that uses an automata theoretic approach. Both the tools Lash and Amaya shall be evaluated and studied in depth for a better understanding of the concepts around their core implementation.

As a main objective, a self written solver based on the Mata automata library will be implemented inside the cvc5 environment. The new implementation shall be compared against Lash and Amaya and also compared against other approaches used to solve LIA problems inside the cv5 solver.

1.2. Organization

The rest of the work presented in this report is organized in the following manner. Section 2 contains the theoretical background needed to understand the work presented in this report. Particularly, section 2.1 contains the basics necessary to comprehend the formal definitions of Presburger arithmetic, while sections 2.2 discuss the topics behind automata theory and the formulation of a Presburger arithmetic problem using an automata theoretic approach. Section 2.3 briefly talks about the current state of the art of SMT solvers and cvc5, the solver used in this project.

Section 3 describes at a high level the architecture of the implemented solver. Section 4 exhibits the experimental evaluations made with the new implementation, showing the comparisons between it and other state of the art solvers in different sets of benchmarks. Finally, section 5 talks about the main contributions of the work done and where and how to access them, while also contains the conclusions made during the development and discussions of future work.

2. Preliminaries

The definitions and formalizations presented in this section were taken from [10, 24, 34, 37, 42]. Some definitions and examples were copied *ipsis literis*.

2.1. Presburger arithmetic

Presburger arithmetic, also know as Linear Integer Arithmetic (LIA), is the first order theory of the natural numbers with addition, formally defined as the first order theory of $(\mathbb{Z}, +, \leq, 0, 1)$.

Scalar multiplication is allowed since it is just syntatic sugar for repetitive addition (ax is $x + \cdots + x$ a times). Moreover, by using negation, multiplication by -1, or subtraction from c by 1, any formula of the form $AX \mathrel{\mathcal{R}} c$, for $\mathcal{R} \in \{=, \leq, \geq, <, >\}$ can be represented in Presburger arithmetic.

Mojżesz Presburger showed the theory was decidable in 1927 [35], and several improvements on his decision procedure were made along the years. However, such approaches weren't able to enumerate solutions neither give a sample vector satisfying the input formula [34]. In 1960, Büchi showed for the first time that finite automata could be used to encode and manipulate fragments of logic [12]. The application of such an idea in Presburger arithmetic was only applied in 1996 by Boudet & Comon [10].

In this project, we consider a LIA formula φ over a finite set of integer variables $\mathbb{X}$ using the following grammar:

$$\varphi_{atom} ::= \vec{a} \cdot \vec{x} = c \mid \vec{a} \cdot \vec{x} \leq c \mid \vec{a} \cdot \vec{x} \equiv_m c \mid \bot$$

$$\varphi ::= \varphi_{atom} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists y(\varphi)$$

where $\vec{x}$ is treated as the set $\mathbb{X} = \{x_1, \ldots, x_n\}$, $\vec{a}$ is a vector of n integer coefficients $(a_1, \ldots, a_n) \in \mathbb{Z}^n$, $c \in \mathbb{Z}$ is a constant, $m \in \mathbb{Z}^+$ is a modulus, and $y \in \mathbb{X}$. The other connectives such as $\top, \rightarrow, \leftrightarrow, \forall, \ldots$ can be constructed in the standard way.

2.2. Automata Theoretic Formulation

The ideia behind this approach is to create a constructive decision procedure for problems in Presburger arithmetic.

Given a formula $\varphi(x_1, \ldots, x_n)$ in the theory, generate a finite automaton $\mathcal{A}_\varphi$ that accepts the set $\{(x_1, \ldots, x_n) \in \mathbb{Z}^n : (x_1, \ldots, x_n) \models \varphi\}$. In simpler words, the goal is to construct a finite automaton $\mathcal{A}_\varphi$ that accepts exactly the set of solutions to the original formula φ.

2.2.1. Finite Automata

A *nondeterministic finite automaton* is a five-tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, \mathcal{I}, \mathcal{F})$, where:

- $\mathcal{Q}$ is a finite set of states
- Σ is an alphabet
- $\delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a transition relation
- $\mathcal{I} \subseteq \mathcal{Q}$ is a set of initial states
- $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states

We define a run of $\mathcal{A}$ over a word $w \in \Sigma^*$ as a sequence of states $\rho = q_0 q_1 \ldots q_n \in \mathcal{Q}^{n+1}$ such that for all $1 \leq i \leq n$ it holds that $(q_{i-1}, a_i, q_i) \in \delta$ and $q_0 \in \mathcal{I}$. The run is *accepting* if $n \geq 1$ and the last state of the run is belongs to $\mathcal{F}$. The language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of words with a accepting run in $\mathcal{A}$.

The automata $\mathcal{A}$ is *deterministic* if $|\mathcal{I}| \leq 1$ and, for all states $q \in \mathcal{Q}$ and symbols $a \in \Sigma$, if $(q, a, p) \in \delta$ and $(q, a, r) \in \delta$, then $p = r$. $\mathcal{A}$ is *complete* if $|\mathcal{I}| \geq 1$ and for all $q \in \mathcal{Q}$ and $a \in \Sigma$, there is at least one state p such that $(q, a, p) \in \delta$. A state is *unreachable* if there is no run from an initial state to it.

Let $q \in Q$ and $S \subseteq Q$ be, respectively, an arbitrary state and a arbitrary subset of states of the automaton, and $\sigma \in \Sigma$ an arbitrary symbol of the automaton alphabet. We define the following functions:

- $pre_\delta(q, \sigma) = \{q' \mid (q', \sigma, q) \in \delta\}$
- $pre_\delta(S, \sigma) = \bigcup_{q \in S} pre_\delta(q), \sigma$
- $post_\delta(q, \sigma) = \{q' \mid (q, \sigma, q') \in \delta\}$
- $post_\delta(S, \sigma) = \bigcup_{q \in S} post_\delta(q, \sigma)$

2.2.2. Classical Automata Based Decision Procedure for Presburger arithmetic

The formal definitions and proofs of the classical decision procedure for LIA formulae using automata are well explained in [10, 20, 24, 34, 37, 42]. In this section, we aim only to present the reader the rough idea of how it works and the information necessary to understand what was implemented. We encourage the reader to check the cited references for more details.

Given a formula φ in Presburger arithmetic theory, a finite automaton $\mathcal{A}_\varphi$ is built that encodes all binary models of φ. This can be done inductively. First, as the base case, a finite automaton $\mathcal{A}_{\varphi_{\text{atom}}}$ is created for each atomic formula φ_{atom} in φ. The procedure constructs an automaton with a finite number of states.

The following algorithms were taken and adapted from [20] and they describe how an automaton is created for atomic formulae with equalities or inequalities.

Algorithm 1 constructs a non-deterministic deterministic automaton for inequalities, and the lack of determinism actually allows for the encoding of solutions over the integers and not only over the naturals. We can see an example of an automaton encoding the solution space of $\varphi : x \leq 4$ over $\mathbb{Z}$ in figure 1, taken from [26].

Algorithm 1 Construction of an NFA enconding solutions of an inequality $\varphi_{\leq}$ over $\mathbb{Z}$

Input: An inequality $\vec{a} \cdot \vec{x} \leq c$ over $\mathbb{Z}$
Output: NFA $\mathcal{A}_{\varphi_{\leq}} = (\mathcal{Q}, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ that encodes the solutions to $\varphi_{\leq}$

$\mathcal{Q}, \delta, \mathcal{F} \leftarrow \emptyset$
$\Sigma \leftarrow \{0,1\}^{|\vec{x}|}$
$\mathcal{I} \leftarrow \{q_c\}$
$\mathcal{W} \leftarrow \{q_c\}$
while $\mathcal{W} \neq \emptyset$ **do**
    $s_k \leftarrow$ pick and remove a state from $\mathcal{W}$
    add s_k to $\mathcal{Q}$
    for every $\sigma \in \Sigma$ **do**
        $v \leftarrow \lfloor \frac{1}{2}(k - \vec{a} \cdot \sigma) \rfloor$
        if $q_v \notin \mathcal{Q}$ **then**
            add q_v to $\mathcal{Q}$ and $\mathcal{W}$
        end if
        add the transition (q_k, σ, q_v) to δ
        $v' \leftarrow \frac{1}{2}(k + \vec{a} \cdot \sigma)$
        if $v' \geq 0$ **then**
            add q_f to $\mathcal{Q}$ and $\mathcal{F}$
            add the transition (q_k, σ, q_f) to σ
        end if
    end for
end while
Return $(\mathcal{Q}, \Sigma, \delta, \mathcal{I}, \mathcal{F})$

Algorithm 2, on the other hand, constructs a NFA encoding the solutions of an equality over $\mathbb{Z}$. There is an example in figure 2 depicting the automaton for the equality $\varphi : x = 4$.

The inductive cases for Boolean connectives are defined in the standard way. Given two formulae φ_1 and φ_2 , the conjunction $\varphi_1 \wedge \varphi_2$ is implemented by taking the intersection of the corresponding automaton, the disjunction $\varphi_1 \vee \varphi_2$ by taking their union, and the negation $\neg\varphi$ by taking the complement of the automaton. Formally, we have:

- $\mathcal{A}_{\neg\varphi}$ encodes $\mathcal{L}(\overline{\mathcal{A}_{\varphi}})$
- $\mathcal{A}_{\varphi_1 \wedge \varphi_2}$ encodes $\mathcal{L}(\mathcal{A}_{\varphi_1}) \cap \mathcal{L}(\mathcal{A}_{\varphi_2})$
- $\mathcal{A}_{\varphi_1 \vee \varphi_2}$ encodes $\mathcal{L}(\mathcal{A}_{\varphi_1}) \cup \mathcal{L}(\mathcal{A}_{\varphi_2})$

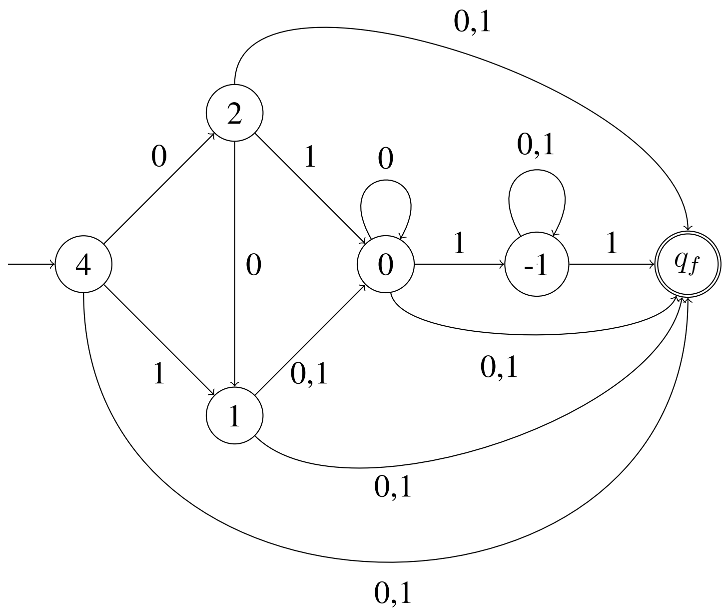


Figure 1. Automaton $\mathcal{A}_\varphi$ for the inequality $\varphi : x \leq 4$

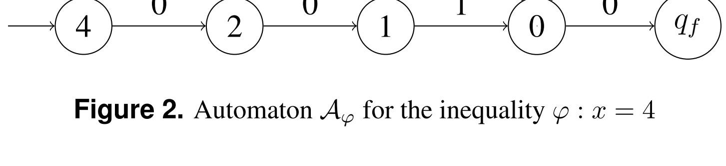


Figure 2. Automaton $\mathcal{A}_\varphi$ for the inequality $\varphi : x = 4$

Algorithm 2 Construction of an NFA enconding solutions of an inequality $\varphi_=$ over $\mathbb{Z}$

Input: An equality $\vec{a} \cdot \vec{x} = c$ over $\mathbb{Z}$

Output: NFA $\mathcal{A}_{\varphi_=} = (\mathcal{Q}, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ that encodes the solutions to $\varphi_=$

```
Q, δ, F ← ∅
Σ ← {0, 1}^{|x⃗|}
I ← {q_c}
W ← {q_c}
while W ≠ ∅ do
    s_k ← pick and remove a state from W
    add s_k to Q
    for every σ ∈ Σ do
        if k − a⃗ · σ is odd then
            continue
        end if
        v ← 1/2 (k − a⃗ · σ)
        if q_v ∉ Q then
            add q_v to Q and W
        end if
        add the transition (q_k, σ, q_v) to δ
        v′ ← 1/2 (k + a⃗ · σ)
        if v′ = 0 then
            add q_f to Q and F
            add the transition (q_k, σ, q_f) to σ
        end if
    end for
end while
Return (Q, Σ, δ, I, F)
```

For quantifiers, the formalization is quite straightforward. The existential quantification $\exists x(\varphi)$ can be solved by projecting away the track of variable x from the automaton. Since there is no direct construction for the universal quantification, we can solve the problem by replacing all of their appearances by the existential one using the equivalence $\forall x(\varphi) \equiv \neg\exists x(\neg\varphi)$.

Projecting away the variable, although, does not solve the entire problem, because the resulting automaton might not encode all the formula solutions anymore. Suppose, for example, the model $\{x = 7 \wedge y = -4\}$, taken from [24]. The shortest word encoding for this model would contain 1110 for the x track and 00011 for the y track. If we remove the x track from the word, we obtain the word 0011, which encodes the model $\{y = -4\}$. However, this is not the shortest encoding of the assignment (the answer would be 001). In simpler terms, we can say that removing a variable track truncates the automaton alphabet, and it might not contain all encoding of solutions anymore.

To solve this, we can use algorithm 3, taken from [26], which augments the automaton structure to solve the issue.

Algorithm 3 *PadClosure*

Input: NFA $\mathcal{A} = (Q, \Sigma, \delta, \mathcal{I}, F)$
Output: NFA $\mathcal{A}' = (Q', \Sigma, \delta', \mathcal{I}, F')$ accepting all enconding of models

$\delta' \leftarrow \delta$, $W \leftarrow \emptyset$
$q_f \leftarrow$ a new state to be added to $\mathcal{A}$ such that $q_f \notin Q$
for $\sigma \in \Sigma$ **do**
    $S \leftarrow \emptyset$, $W \leftarrow pre_\delta(F, \sigma)$
    while $W \neq \emptyset$ **do**
        $q \leftarrow$ pick and remove an element from W
        add q to S
        for $q' \in pre_\delta(q, \sigma)$ **do**
            if $q' \notin S$ **then**
                add q' to W
            end if
        end for
    end while
    for $q \in S$ **do**
        if $post_\delta(q, \sigma) \cap F = \emptyset$ **then**
            add (q, σ, q_f) to δ'
        end if
    end for
end for
if $\delta = \delta'$ **then**
    return $(Q, \Sigma, \delta, \mathcal{I}, F)$
else
    return $(Q \cup \{q_f\}, \Sigma, \delta', \mathcal{I}, F \cup \{q_f\})$
end if

2.3. SMT Solvers and cvc5

SMT solvers are tools built with the goal of solving SMT problems. There are many state of the art solvers out there, each of them with their respective particularities and design choices. In this project we are goind to use the cvc5 [2] solver, which is an open-source automatic theorem prover, written entirely in C++ and released under an open-source software license. All the code and build instructions can be found in the tool's GitHub repository.

3. Architecture

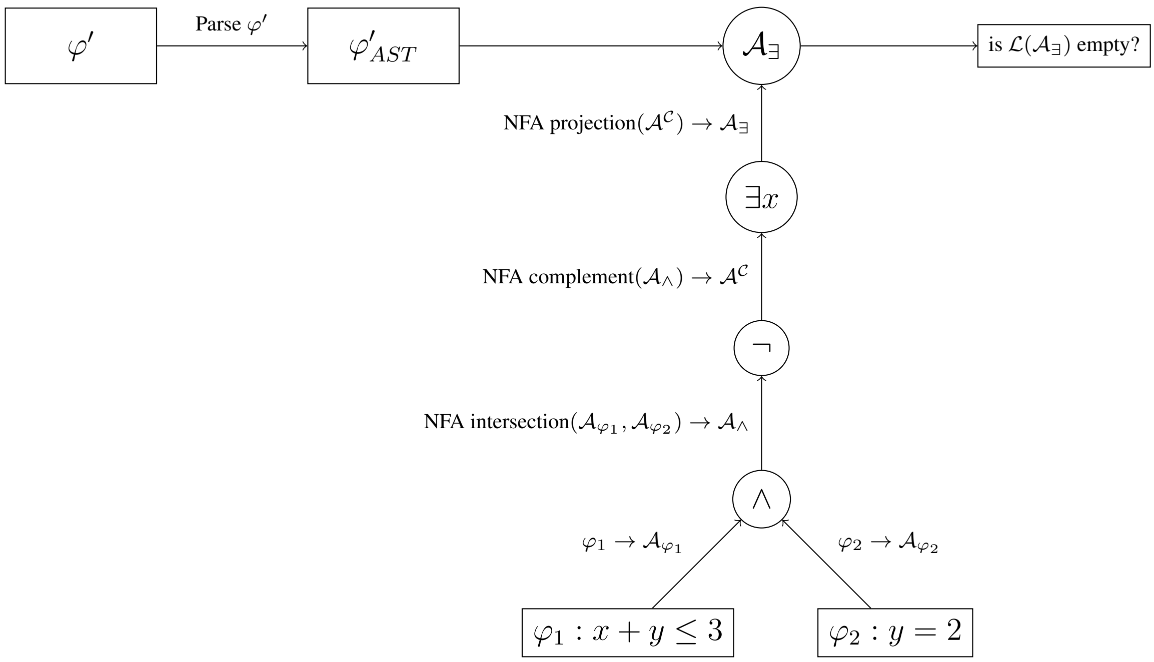
As aforementioned, the solver was implemented over the cvc5 environment, making usage of several of it's utilities such as the SMT-LIB [4] language parser. For the current state of the solver, it's assumed that the input was preprocessed by the Amaya [26] tool, which applies a lot of the formula rewriting described in [24]. After preprocessing, the input formula is fed to the cvc5 binary, which will decide it's satisfiability.

Inside cvc5, the solver is implemented as a preprocessing step. In the step, the automaton is constructed inductively, in a bottom-up manner, as described in section 2.2.2. Then, the final automaton goes through a emptiness check. If the language of the automaton is empty, the formula is `unsat`, and `sat` otherwise. The choice of implementing it that way is mainly to make cvc5 leave quantifiers untouched, and let the automaton solver deal with the entire formula.

Below there is a diagram ilustrating the general process of how the implemented solver deals with an example Presburger arithmetic formula $\varphi : \exists x(\neg(x+y \leq 3 \wedge y = 2))$.

$$\boxed{\varphi} \xrightarrow{\text{Amaya preprocessing}} \boxed{\varphi'} \xrightarrow{\text{cvc5}} \boxed{\texttt{sat}}$$

Inside the cvc5 procedure step, we have:



4. Benchmarks

For the evaluation of the implemented tool, two different types of benchmark were utilized, based on the benchmarks presented in [26]. First, our implementation was compared against cvc5's default linear integer arithmetic solver in a set of formulae from CAV 2009 datasets. Secondly, we compared our solver against Amaya, Lash, Z3 and cvc5 on the solving of formulae representing the Frobenius coin problem [38] with two variables, available at Amaya's GitHub repository.

The experiments were made in a machine with the following configuration:

Property	**Details**
OS	Ubuntu 24.04.1 LTS x86_64
Host	MS-7D20 1.0
Kernel	6.8.0-51-generic
CPU	11th Gen Intel i5-11400F (12) @ 4.400GHz
GPU	NVIDIA GeForce GTX 1660 SUPER
Memory	16GB

Table 1. System information

4.1. CAV 2009 benchmarks

CAV (International Conference on Computer-Aided Verification) is an academic conference on the theory and practice of computer-aided verification, one of the most important ones in the context of SMT solving. The 2009 conference made available a set of benchmarks for LIA problems that suit very well our needs for benchmarking. It can be found at this link, where lots of other SMT-COMP benchmarks are hosted.

The dataset contains quantifier free formulae that vary in the size of the coefficients and in the number of variables. Firstly, we tested our solver, as well other solvers based on automata, in the formulae varying the size of the coefficients. As the order of magnitude of the coefficients rises, our solver was not even able to finish before the process was terminated by the OS because of memory usage. This has a lot of reasons, one of which being the fact that out implementation doesn't yet support the optimizations needed for the construction of automata with less states than needed. We use a bottom up naive approach, unlike Amaya, who uses a top-down automaton construction that allows a automaton construction with a considerably smaller number of states created during computation [27].

We also need to refer to the size of the coefficients, since this causes a huge impact in the number of states created during construction of the automaton [18]. The decision procedure based on automata becomes infeasible, even for simple formulae, when such types of constraints appear. This issue can be confronted by the optimizations described in [24].

In second place, we also evaluated our solver in the benchmarks where the number of variables vary. As aforementioned, the whole decision procedure has a exponential blowup regarding the number of variables in the input formula, so we expected to see this behavior in the tests, and we did. Our solver couldn't handle any of the benchmarks with more than 15 variables, while it was able to solve a big part of the formulae with 10 variables.

The results found during this step of benchmarking actually helped to confirm that the automata-based approach is not particularly efficient for handling quantifier free formulae, but instead it's advantage is in dealing with formulae with a large amount of quantifiers, as we will see in the next subsection.

4.2. Frobenius coin problem

The Frobenius coin problem is a famous algebra problem named after the German mathematician Fernidand Georg Frobenius. Althought it is a problem with a notorious theoretical importance [1], it's fame also comes from a joke involving Chicken McNuggets.

The problem asks what is the largest number n you can get such that n is not the result of the sum of elements from a set $\mathcal{S}$ of coprime numbers, which we will refer as coins. Such a number is called the *Frobenius number* of set $\mathcal{S}$. If the amount of coins is 1, the problem is trivial. If there are two coin denominations, a_1 and a_2, the *Frobenius number* can be found with the formula $a_1 a_2 - a_1 - a_2$, found by mathematician James Joseph Sylvester in 1882. For more than two coins the problem starts to get a little tricky, and no general formula is known. The general Frobenius coin problem, with an arbitrary number of coins, is know to be NP-Hard [7]. However, if the number of coins is fixed, there is a polynomial time algorithm for finding the solution [29].

The problem can be written using the following formula, taken from [26], where f is the Frobenius number and $\vec{w}$ is a vector of setwise coprime numbers:

$$\text{Frob}(f, \vec{w}) \triangleq \forall \vec{n} \in \mathbb{N}^{|\vec{w}|} : (f \neq \vec{w} \cdot \vec{n}) \wedge (\forall f' \in \mathbb{N} : ((\forall \vec{n}' \in \mathbb{N}^{|\vec{w}|}(f' \neq \vec{n}' \cdot \vec{w})) \rightarrow f' \leq f))$$

As we can see, the problem formulation contains a non-negligible amount of quantifiers, so it is a good way of testing the capacity of our approach for solving quantifier-heavy formulae. The benchmark used for evaluating the implemented solution with the state of the art is the same used in [26]. It consists of a set of instances of $\mathrm{Frob}(n, \vec{w})$ with two coins with consecutive primes denominations. Figure 3 shows that our solver vastly outperforms cvc5 and Z3 when solving this set of problems. We've established a time limit of 120 seconds for the solvers.

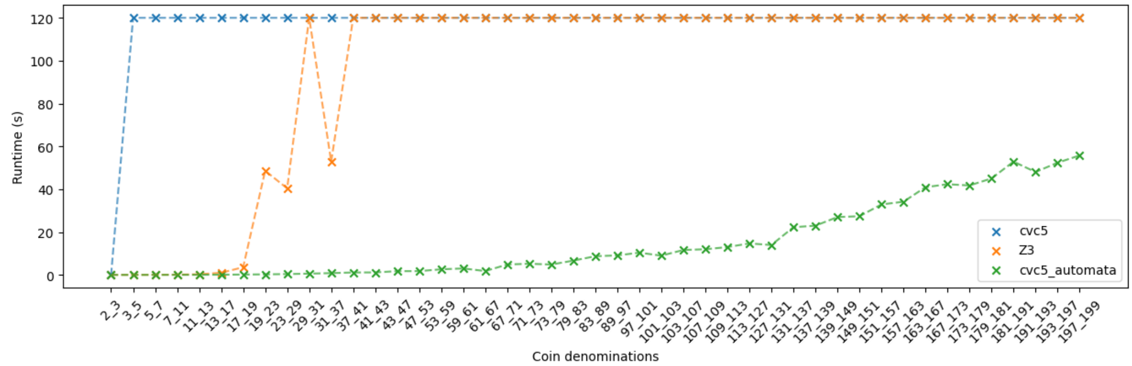


Figure 3. Frobenius Coin problem benchmark

Against other automata-based decision procedures, we can see, in figure 4, that out solver keeps up with Amaya in the first half, but it gets slower as the coin denominations value increases. The reason is because, as discussed before, the size of the automaton is directly impacted by the values of the coefficients present in the formula. Amaya implements heuristics and optimizations already thought to mitigate this kind of problem and our implementation doesn't yet.

Lash on the other hand starts faster than any other solver but as the coin denominations values starts to increase, it's behavior begins to be very slow and fraught.

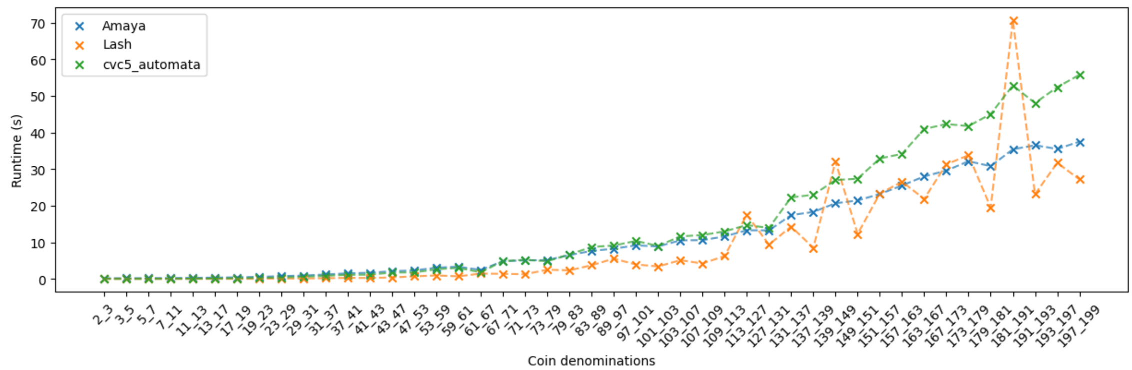


Figure 4. Frobenius Coin problem benchmark

5. Conclusion and future work

In this project, we managed to add to the cvc5 environment an automata based approach for solving Presburger arithmetic formulae. The implementation used the Mata automata library as a backend for handling automata operations such as the intersection and union, for example. The fork of cvc5 with the implementation of the new approach can be found in this GitHub repository, with instructions on how to build the project and where to find the benchmarks used. Since the time available for the implementation of the project was short, the main focus was to create a working MVP, which was accomplished, but it's performance is not up to the level of Amaya, as shown in 4. For future work, we aim to study in more depth the core of Amaya's source code to understand how can we improved our code to match up with the state of the art solvers in the usage of automata theory for solving LIA problems.

In addition to the LIA solver implemented, another contribution of this monograph was the integration of the Mata library in the cvc5 ecossystem. Mata is an automata library that offers a combination of speed and simplicity. Handling finite automata is a hard task and Mata can be used to mitigate this problem. It can be used by cvc5 for solving other types of problems, such as string theory formulae, the same way it was used in Z3-Noodler [14], for example.

The implemented tool opens a range of possibilities of optimizations, like the ones described in [24]. The preprocessing passes, for example, shall be implemented in cvc5 and evaluated in future work, in order to not rely on Amaya's formulae rewriting implementation.

References

[1] JL Ramırez Alfonsın. The diophantine frobenius problem. *Forschungsintitut für Diskrete Mathematik, Bonn, Report N*, page 00893, 2000.

[2] Haniel Barbosa, Clark Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, et al. cvc5: A versatile and industrial-strength smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 415–442. Springer, 2022.

[3] Clark Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of satisfiability*, pages 1267–1329. IOS Press, 2021.

[4] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, page 14, 2010.

[5] Clark Barrett, Cesare Tinelli, Haniel Barbosa, Aina Niemetz, Mathias Preiner, Andrew Reynolds, and Yoni Zohar. Satisfiability modulo theories: A beginner's tutorial.

[6] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. (lia)-model evolution with linear integer arithmetic constraints. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 258–273. Springer, 2008.

[7] Dale Beihoffer, Jemimah Hendry, Albert Nijenhuis, and Stan Wagon. Faster algorithms for frobenius numbers. *the electronic journal of combinatorics*, 12(1):R27, 2005.

[8] Chris Bernhardt. *Turing's vision: the birth of computer science*. MIT Press, 2016.

[9] Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. An effective decision procedure for linear arithmetic over the integers and reals. *ACM Transactions on Computational Logic (TOCL)*, 6(3):614–633, 2005.

[10] Alexandre Boudet and Hubert Comon. Diophantine equations, presburger arithmetic and finite automata. In *Trees in Algebra and Programming—CAAP'96: 21st International Colloquium Linköping, Sweden, April 22–24, 1996 Proceedings 21*, pages 30–43. Springer, 1996.

[11] Martin Bromberger, Thomas Sturm, and Christoph Weidenbach. Linear integer arithmetic revisited. In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 623–637. Springer, 2015.

[12] J Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6), 1960.

[13] Shaowei Cai, Bohan Li, and Xindi Zhang. Local search for smt on linear integer arithmetic. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification*, pages 227–248, Cham, 2022. Springer International Publishing.

[14] Yu-Fang Chen, David Chocholatỳ, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Z3-noodler: An automata-based string solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 24–33. Springer, 2024.

[15] David Chocholatỳ, Tomáš Fiedor, Vojtěch Havlena, Lukáš Holík, Martin Hruška, Ondřej Lengál, and Juraj Síč. Mata: A fast and simple finite automata library. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 130–151. Springer, 2024.

[16] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[17] Haoran Ding, Zhaoguo Wang, Yicun Yang, Dexin Zhang, Zhenglin Xu, Haibo Chen, Ruzica Piskac, and Jinyang Li. Proving query equivalence using linear integer arithmetic. *Proceedings of the ACM on Management of Data*, 1(4):1–26, 2023.

[18] Antoine Durand-Gasselin and Peter Habermehl. On the use of non-deterministic automata for presburger arithmetic. In *International Conference on Concurrency Theory*, pages 373–387. Springer, 2010.

[19] Herbert B Enderton. *A mathematical introduction to logic*. Elsevier, 2001.

[20] Javier Esparza and Michael Blondin. *Automata theory: An algorithmic approach*. MIT Press, 2023.

[21] Tomáš Fiedor. Automata in decision procedures and performance analysis. 2019.

[22] Michael J Fischer and Michael O Rabin. Super-exponential complexity of presburger arithmetic. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, pages 122–135. Springer, 1998.

[23] Christoph Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.

[24] Peter Habermehl, Vojtěch Havlena, Michal Hečko, Lukáš Holík, and Ondřej Lengál. Algebraic reasoning meets automata in solving linear integer arithmetic. In *International Conference on Computer Aided Verification*, pages 42–67. Springer, 2024.

[25] Md Mohedul Hasan and Md Rajib Arefin. Application of linear programming in scheduling problem. *The Dhaka University Journal of Science*, 65(2):145–150, 2017.

[26] Michal Hečko. An automata-based decision procedure. Bachelor's thesis, Brno University of Technology, Brno, Czech Republic, 2022. Supervisor: Ing. Ondřej Lengál, Ph.D.

[27] Michal Hečko. Deciding logic with automata / rozhodování logiky pomocí automatů. Master's thesis, Brno University of Technology, Faculty of Information Technology, Brno, Czech Republic, 2024.

[28] Dustin Hütter. *SMT solving for linear integer arithmetic*. PhD thesis, Bachelor's thesis, RWTH Aachen University, 2014.

[29] Ravi Kannan. Lattice translates of a polytope and the frobenius problem. *Combinatorica*, 12(2):161–177, 1992.

[30] Felix Christopher Klaedtke. *Automata-based decision procedures for weak arithmetics.* PhD thesis, University of Freiburg, Freiburg im Breisgau, Germany, 2004.

[31] Fritz Kliemann. Deciding presburger arithmetic using automata theory/submitted by fritz kliemann, bsc. 2018.

[32] Dexter C Kozen. *Automata and computability*. Springer Science & Business Media, 2012.

[33] Jérôme Leroux. A polynomial time presburger criterion and synthesis for number decision diagrams. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*, pages 147–156. IEEE, 2005.

[34] Mia Minnes. Mixed and integer linear programming using automata techniques, 2018.

[35] Mojzesz Presburger. Uber die vollstandigkeiteines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt. In *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs*, 1929.

[36] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.
[37] J Richard. Büchi. on a decision method in restricted second-order arithmetic. In *Proceedings of the International Congress on Logic, Math, and Philosophy of Science. Standford University Press*, 1962.
[38] Jeffrey Shallit. The frobenius problem and its generalizations. In *International Conference on Developments in Language Theory*, pages 72–83. Springer, 2008.
[39] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.
[40] Tom Van Dijk and Jaco Van De Pol. Sylvan: Multi-core decision diagrams. In *Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21*, pages 677–691. Springer, 2015.
[41] Pierre Wolper and Bernard Boigelot. An automata-theoretic approach to presburger arithmetic constraints. In *International Static Analysis Symposium*, pages 21–32. Springer, 1995.
[42] Pierre Wolper and Bernard Boigelot. On the construction of automata from linear arithmetic constraints. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 1–19. Springer, 2000.