BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2022
http://excel.fit.vutbr.cz

An Automata-Based Decision Procedure for Presburger Arithmetic

Michal Hečko*

Abstract
Presburger arithmetics (PrA) is a decidable, first-order theory of natural numbers, with applications in many areas in formal verification of software properties. SMT-solvers — tools implementing various algorithmic approaches to deciding whether a formula has a solution — play a crucial role in formal verification. In this paper, we document building a novel automatic SMT solver for PrA based on finite automata — an approach that no SMT solver currently employs. We provide an overview of challenges and their solutions arising from the complexity of such a tool, including results from the concluded experiments already showing promise of this alternative approach. We also present identified problems where the performance of the automata-based procedure struggles, which present open research opportunities.

Keywords: SMT Solver — Presburger arithmetic — Finite automata

Supplementary Material: GitHub repository

*xhecko02@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

As mankind advances, it increasingly relies on the use of computers to offload some of the tasks previously requiring human intellect to autonomous machines. This growing penetration of information technologies into everyday life creates the necessity to build safe and reliable software controlling those devices, bringing attention to the role of formal verification of software.

One of the core tools used in program and hardware verification is a Satisfiability Modulo Theories (SMT) solver. The purpose of this tool is to automatically deduce the existence of a solution to a given formula, reasoning about the properties of examined system. An SMT solver as such is a versatile tool with applications not limited only to verification, but also in various other areas of modern computer science such as compiler optimization techniques or automated theorem proving, further highlighting its importance.

One of the prominent theories used in the input formulae of SMT solvers is Presburger Arithmetic (PrA) [1]. This first-order theory provides a formal basis for describing a system using linear integer arithmetic constraints. The signature of this theory of natural numbers contains only a symbol for zero, a symbol for a successor and a symbol for addition with corresponding axioms. Not having a notation for multiplication implies a limitation of its expressive power, but it also allows the theory to remain decidable. This property states that it is possible to deduce algorithmically whether a formula has a model in a finite number of steps. Over time there have been numerous such algorithms — *decision procedures* — developed, approaching the problem of determining the existence of a model from different perspectives. A lot of current research focuses on developing various heuristics improving the performance of these procedures e.g. [2], extending what we can decide automatically.

One of the younger decision procedures for PrA is based on ideas introduced by Büchi in 1960 [3]. Büchi created a decision procedure based on the formal model of finite automata, in which parts of the input formula are represented as automata and these are combined using operations on automata according to logical connectives between the represented parts of the input formula, copying the formula structure. For an automaton to represent a formula, it must exactly

accept all solutions of the formula encoded using some chosen encoding. Büchi developed this decision procedure to show that a different theory (a second-order theory of natural numbers called S1S) is decidable, and it was not until 1996 when Boudet & Comon [4] modified it to be used to decide PrA. To the best of our knowledge, no decision procedure for PrA based on this approach has been implemented in any SMT solver yet. Moreover, the underlying formal model of finite automata is a vivid research area with advancements such as new automata models [5], or more efficient language inclusion checking of nondeterministic finite automata [6]. The lack of any SMT solver implementing the automata-based decision procedure for PrA means that advancements in the automata field have not yet been applied and evaluated within the context of deciding PrA. The relative youth and the lack of an SMT solver implementing this procedure mean that an automata-based SMT solver for PrA is still an unexplored field.

1.1 Contribution of the paper

We have succeeded in building a robust automata-based SMT solver for PrA. The complexity of this tool created numerous challenges that had to be solved to achieve a functional implementation. These challenges include scalability problems due to the exponential growth of time complexity wrt. the number of variables in the input formula, or representing Boolean variables as automata.

The implemented SMT solver was designed to provide a foundation for future research in the practical applications of automata in the context of deciding PrA. Therefore, our solver supports a standardized input language, allowing us to compare the automata-based procedure to the state-of-the-art solvers. Our early experiments presented in this work already identified problems where automata vastly outperform other approaches utilized by the state-of-the-art SMT solvers such as Z3 or CVC4. Having a functional automata-based solver allowed us also to identify performance bottlenecks of the underlying decision procedure that present open research opportunities.

2. Description of an Automata-Based Decision Procedure

Presburger arithmetic is a first-order theory of integers $Th(Z, +, 0, 1, <)$. The theory does not contain a symbol for multiplication, however multiplication of a variable by a constant is possible, as it is a shorthand for summation:

$$nx \stackrel{def}{=} \underbrace{x + \cdots + x}_{n}$$

We say that the PrA formula ψ is *atomic* iff it has the form $\psi : \vec{a} \cdot \vec{x} \sim c$ where $\vec{a}$ is a vector of constant variable coefficients, $\vec{x}$ is a vector of variables, c is a constant, and $\sim$ is one of $\{\leq, <, =, \equiv_M\}$ where M is a constant. An example of an atomic formula can be seen in Equation 1.

$$3x + 4y - 10z \leq 3 \tag{1}$$

A *finite automaton* is a 5-tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$, where:

- Q is a finite non-empty set of states,
- Σ is a finite non-empty set of symbols, called an alphabet,
- δ: $Q \times \Sigma \to 2^Q$ is the transition function,
- $Q_0 \subseteq Q$ is the set of initial states,
- $F \subseteq Q$ is the set of final states.

Let $w \in \Sigma^*$ be a string of alphabet symbols called a *word*, $|w|$ denotes the length of this word and let w_i denote the i-th symbol of the word w for every $1 \leq i \leq |w|$. A *run* of the automaton $\mathcal{A}$ over the word $w \in \Sigma^*$ is a sequence of its states $q_0, q_1, \cdots, q_{|w|}$ that satisfies $q_i \in \delta(q_{i-1}, w_i)$ for every $1 \leq i \leq |w|$ and $q_0 \in Q_0$. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$, if there exists a run $q_0, \cdots, q_{|w|}$ of $\mathcal{A}$ over this word for which $q_{|w|} \in F$. The *language* of an automaton $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is the set of all words accepted by $\mathcal{A}$.

A formula ψ is represented via an equivalent automaton $\mathcal{A}_\psi$ iff $\mathcal{L}(\mathcal{A}_\psi)$ contains exactly the solutions of ψ encoded using some chosen encoding.

2.1 *LSBF* encoding

To encode models of PrA formulae, we represent their solution space in a binary Least Significant Bit First (LSBF) encoding. In contrast to non-binary encodings, LSBF allows for more compact automaton transition relations. LSBF also allows for a symbolic representation of the transition symbols using well-known formalisms such as Binary Decision Diagrams [7]. LSBF is based on two's complement — a binary representation of signed integers in which every number x is represented by a bit vector $b_0 \ldots b_{N-1}$ such that the Equation 2 holds.

$$x = -2^{N-1} a_{N-1} + \sum_{i=0}^{N-2} 2^i a_i \tag{2}$$

Equation 3 shows a two's complement representation of positive and negative numbers. However, as

automata are synchronous machines, we require that all tracks have the same length. Extending a two's complement representation by repeating the a_{N-1} bit — the sign bit — does not change the represented value, making it possible for all variable tracks to have the same length as shown in Equation 4. As the name suggests, the LSBF reads the numbers in reverse order, placing the least significant bits first on the input tape. Equation 5 illustrates the LSBF encoding with track values split into individual alphabet symbols as read by an automaton.

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -3_{10} \\ 10_{10} \end{pmatrix} = \begin{pmatrix} 101_2 \\ 01010_2 \end{pmatrix} \tag{3}$$

$$= \begin{pmatrix} 11101_2 \\ 01010_2 \end{pmatrix} \tag{4}$$

$$= \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)_{LSBF} \tag{5}$$

2.2 The Mechanism of an Automata-Based Decision Procedure

Logical connectives combining PrA formulae map naturally to operations on regular languages and the corresponding finite automata. Let φ and ψ denote two formulae in Presburger arithmetic, $\mathcal{A}_\varphi$ and $\mathcal{A}_\psi$ be automata encoding those formulae, and $\mathcal{L}(\mathcal{A})$ denote the language of an automaton $\mathcal{A}$. Then the construction proceeds as follows:

- *Negation* $\neg\psi \rightsquigarrow \mathcal{A}_\psi^C$, where $\mathcal{A}_\psi^C$ encodes $L(\overline{\mathcal{A}_\psi})$ (Automaton complement)
- *Conjunction* $\varphi \wedge \psi \rightsquigarrow \mathcal{L}(\mathcal{A}_\varphi) \cap \mathcal{L}(\mathcal{A}_\psi)$
- *Disjunction* $\varphi \vee \psi \rightsquigarrow \mathcal{L}(\mathcal{A}_\varphi) \cup \mathcal{L}(\mathcal{A}_\psi)$

Adding *padding* — extending the bit vector representing some variable assignment by the sign bit — does not change the encoded value, therefore, every variable assignment has an infinite number of corresponding bit vectors. Automata used during the decision procedure must accept all possible encodings, otherwise the same variable assignment, but with different encoding would be accepted by the automaton $\mathcal{A}_\psi$ representing solutions of ψ and the automaton $\mathcal{A}_{\neg\psi}$ corresponding to $\neg\psi$.

The existential quantifier $\exists x(\psi)$ maps to removing the track for variable x in the automaton $\mathcal{A}_\psi$ representing solutions of ψ. The track removal results in a nondeterministic automaton $\mathcal{A}_{\exists x(\psi)}$ as the transitions over symbols differing only in bits on the track for the variable x no longer differ. Removing a track expresses what the existential quantifier does in terms of automata language — a variable assignment of the remaining unbound variables is accepted by $\mathcal{A}_{\exists x(\psi)}$ only if there was an accepting run of $\mathcal{A}_\psi$ over the same assignment together with some assignment of x. As there is no direct operation on automata corresponding to the universal quantifier $\forall x$, universal quantifiers must be rewritten in terms of existential quantifiers and negation according to the law: $\forall x\varphi(x) \Leftrightarrow \neg\exists x(\neg\varphi(x))$.

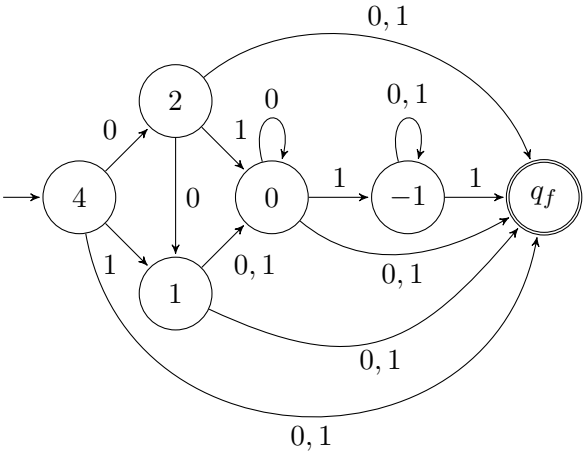

Figure 2. Automaton A_φ for the inequality $\varphi : x \leq 4$ over $\mathbb{Z}$

The automata-based decision procedure works in a bottom-up manner, starting by constructing automata encoding atomic constrains using the $IneqToNFA$ algorithm. The $IneqToNFA(\alpha)$ construction creates an automaton $\mathcal{A}_\alpha$ accepting the solutions of $\alpha : \vec{a} \cdot \vec{x} \leq c$, where $\vec{x}$ is a vector of variables, $\vec{a} \in \mathbb{Z}^{|\vec{x}|}$ is a vector of variable coefficients, and $c \in \mathbb{Z}$ is a constant. The resulting automaton $\mathcal{A}_\alpha$ has every state q_i labeled by an integer c_i such that the language of every state q_i are the solutions of $\vec{a} \cdot \vec{x} \leq c_i$. The only exception to this labeling schema is the state q_f, serving the purpose of nondeterministically guessing that the symbol read from any state is the last one, and therefore, it presents the sign bit vector. Only after reading the signs can we decide whether the read input is a model, hence, q_f is the only accepting state. An example of an automaton constructed by the $IneqToNFA$ procedure is portrayed in Figure 2.

After constructing automata for atomic constraints, the procedure continues up the syntax tree of the formula, combining the intermediate automata according to the description above. After the entire input formula ψ is processed, the decision procedure has constructed an automaton $\mathcal{A}_\psi$ with $\mathcal{L}(\mathcal{A}_\psi)$ containing all models of ψ. Therefore, ψ has a model iff $\mathcal{L}(A) \neq \emptyset$. The entire decision procedure is illustrated in Figure 1.

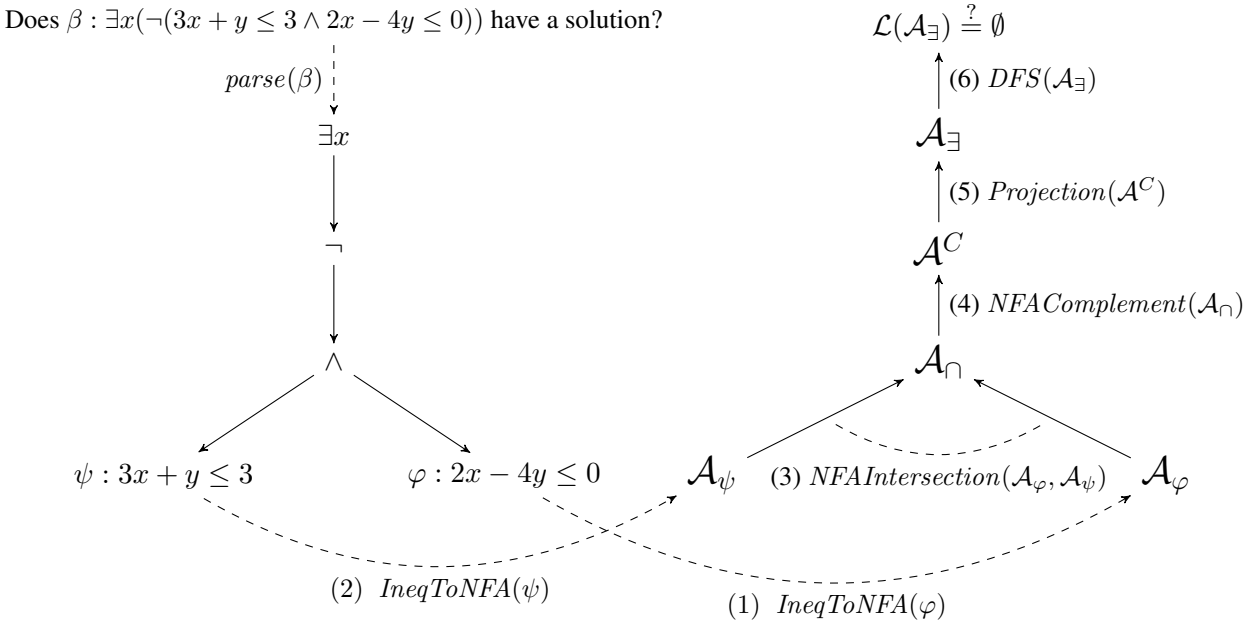

Figure 1. An illustration of the decision procedure for a formula β: $\exists x(\neg(3x + y \leq 3 \wedge 2x - 4y \leq 0))$.

3. Implementation of the automata-based decision procedure for PrA

Our solver is implemented using the Python programming language. The choice was driven by the desire to be able to modify the implementation quickly and to perform experiments with as little friction as possible. Python presented itself as a great choice due to its dynamic nature and the vast library ecosystem. Naturally, this decision impacts the performance, however, our aim was not to best the state-of-the-art SMT solvers, but rather to study the decision procedure and identify its strengths and weaknesses.

One of the early design decisions was to support a reasonable subset of SMT-LIB [1] — a standardized Lisp-like language for encoding input formulae. Technically, any SMT solver accepting this language is a Lisp interpreter with a unique evaluation strategy, including ours. Therefore, our implementation has to deal with various aspects of implementing an interpreter such as variable scopes or input preprocessing removing syntactic sugar such as macros.

The core of the implementation is an automata library implementing all the algorithms used throughout the decision procedure, ranging from necessary algorithms such as procedures for computing the intersection of given automata to optional algorithms for experimentation purposes such as various automata minimization procedures.

Initially, automata were represented in a fashion similar to the formal definition. Due to practical reasons, some restrictions were made during later development, such as limiting the automaton states to only plain integers instead of any type or disallowing automata to have distinct alphabets during the decision procedure. Automata constructions typically produce automata with states carrying semantical information, tying the states back to the construction inputs. For example, the states of the output automaton of the intersection procedure are tuples containing one state from each of the input automata. However, by restricting the state types to only integers, the states can no longer reflect their semantics. The lost state semantics can be optionally gained back as our automata library can track the state semantics in the automaton metadata.

One of the crucial algorithms that needed to be designed from scratch was an algorithm augmenting the automaton structure after a variable track has been projected away as a consequence of an existential quantifier. Removing a variable track changes the sign symbol, and as the automaton does not reflect this change, there might be solution encodings with a certain number of sign bit repetitions that are not accepted. Compared to automata-based decision procedures for other theories, such as WS1S, every alphabet symbol can be a sign symbol, further complicating the problem.

After building a prototype capable of deciding simple formulae, we encountered scalability issues caused by the increased number of variables present in the input formula when attempting to solve some of the more advanced benchmarks. These issues arise due to the design of classical automata algorithms relying on iterating over the entire alphabet that grows exponentially wrt. the number of variables used. We solved

[1] The standard is available at https://smtlib.cs.uiowa.edu/language.shtml

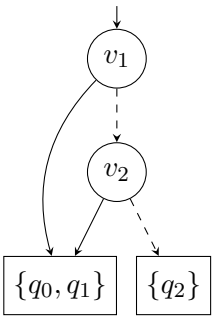

Figure 3. An MTBDD representing the function $f(v_1, v_2) = \{(0,0) \mapsto \{q_2\}, (0,1) \mapsto \{q_0, q_1\}, (1,?) \mapsto \{q_0, q_1\}\}$ where the ? symbol stands for a don't-care bit.

this problem by using Multi-Terminal BDDs [8] — a BDD variant that allows leaves to have arbitrary values instead of one and zero — to store and manipulate the transition relation in a compressed fashion. An example of an MTBDD can be seen in Figure 3. The implementation relies on Sylvan [9] — a library providing a performant MTBDD implementation. As Sylvan is written in the C programming language, we had to create a C library providing Sylvan with custom MTBDD leaves and operators manipulating MTBDDs containing these leaves. To fully utilize the benefits MTBDDs provide, we had to redesign all needed classical automata algorithms in terms of MTBDDs, avoiding uncompressing the symbolic representation they provide. A custom wrapper had to be written that abstracts away the low-level details such as memory management and data serialization, allowing interaction with the C library from Python. Our solver, therefore, provides two execution backends the user can choose from: the original one storing symbols explicitly, enabling easy experimentation, and the high-performance MTBDD-based one.

4. Experimental evaluation

We performed numerous synthetic tests when evaluating the performance improvements of the MTBDD backend. Speedups were observed with all automata operations, especially with the intersection procedure and the determinization procedure. The runtime improvements of the determinization procedure can be seen in Figure 4.

Our solver was also compared with the state-of-the-art SMT solvers Z3 and CVC4 on deciding the Frobenius coin problem instances. The Frobenius coin problem [10] is a famous mathematical problem that can be formulated in the following fashion: What is the largest amount not obtainable from given coins of certain denominations? The precise formulation is given by the following formula where $\vec{w}$ is the vector

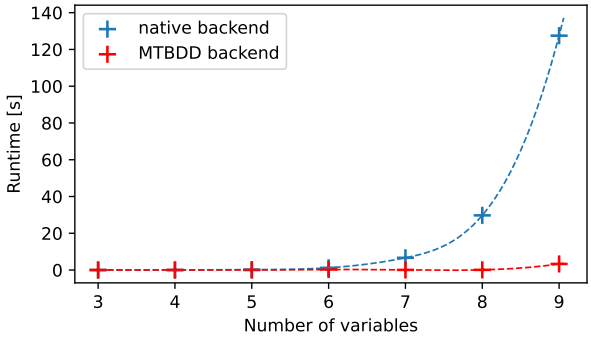

Figure 4. Comparison of determinization speed on randomly generated automata

of coin denominations, m is the number of coins and f is the solution:

$$Frob(f) \colon \forall \vec{n} \in \mathbb{N}_0^m (f \neq \vec{w} \cdot \vec{n}) \wedge (\forall f' \in \mathbb{N}_0 (\forall \vec{n'} \in \mathbb{N}_0^m (f' \neq \vec{n'} \cdot \vec{w})) \rightarrow f' \leq f)$$

The coin problem is not limited only to currency, and it also appears in some areas of computer science, such as the analysis of P-systems [11]. The results presented in Figure 5 show that the automata-based procedure vastly outperforms Z3.

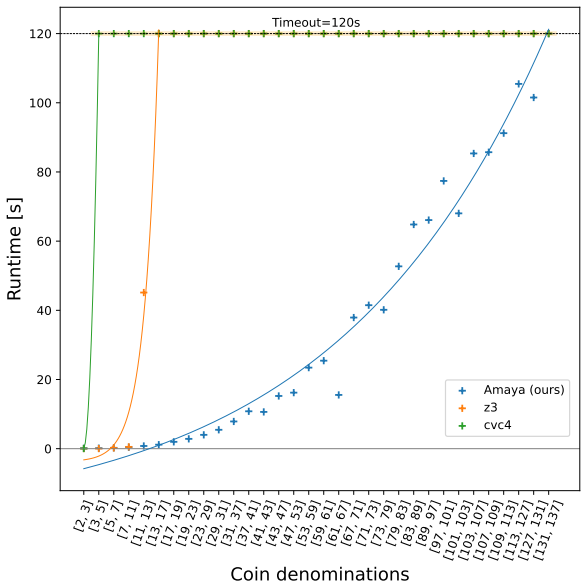

Figure 5. Comparison of runtime between our solver and the state-of-the-art solvers Z3 and CVC4 on instances of the Frobenius coin problem with two coins and gradually increasing coin denominations.

Experimentation also identified interesting problems with automata for atomic constraints having too many states. More specifically, the benchmarks (originating from program verification) contain atomic formulae of the following form $\vec{a} \cdot \vec{x} \equiv_M 0$ where M is a constant in the order of hundreds of thousands. The automaton for such a formula has M states, causing the procedure to become prohibitively

expensive. We are currently researching the possibility to use the cyclic nature of modulo to represent such atomic constraints symbolically.

5. Conclusions

We have succeeded in creating an SMT solver for linear arithmetic based on the formal model of finite automata — an approach that no other SMT solver employs. By building this tool, we have laid the necessary foundation required for future research in the practical applications of automata in the context of deciding Presbuger arithmetic. The created solver supports a standardized input language, allowing easy comparison of our solver to the state-of-the-art solvers implementing different approaches. The ability to compare the different approaches allows us to identify problems where automata present a more performant decision procedure, pushing the limits of what we can decide automatically. Our experimentation already yielded very positive results in favor of automata-based procedure when deciding the Frobenius coin problem.

Our experiments also found benchmarks in which the state-of-the-art solvers perform poorly, and therefore, these benchmarks present an opportunity for automata to provide a more performant solution. These benchmarks include atomic constraints with modulo terms having moduli in order of several hundreds of thousands, causing automata for these constrains to have unfeasibly many states. In our future work, we would like to focus on representing parts of these automata symbolically by utilizing the cyclic nature of modular arithmetic.

Acknowledgements

I would like to thank my supervisor Ing. Ondřej Lengál, Ph.D., for his patience and guidance. I also wish to express my thanks to doc. Mgr. Lukáš Holík, Ph.D. and Ing. Vojtěch Havlena, Ph.D. for providing me with inspiration and many constructive ideas. Finally, my gratitude belongs to Michaela Mesárošová for her tremendous support.

References

[1] M Presburger. Uber die vollständigkeit eines gewissen systems der arithmetik der ganzen zahlen, in dem die addition als einzige operation hervortritt, cempter rendus du 1. In *Congres des Mathematiciens des pays Slaves, Warszawa*, volume 129, pages 92–101, 1929.

[2] Dmitry Chistikov, Christoph Haase, and Alessio Mansutti. Presburger arithmetic with threshold counting quantifiers is easy. *CoRR*, abs/2103.05087, 2021.

[3] J. Richakd Büchi. Weak second-order arithmetic and finite automata. In Saunders Mac Lane and Dirk Siefkes, editors, *The Collected Works of J. Richard Büchi*, pages 398–424, New York, NY, 1990. Springer New York.

[4] Alexandre Boudet and Hubert Comon. Diophantine equations, presburger arithmetic and finite automata. In Hélène Kirchner, editor, *Trees in Algebra and Programming — CAAP '96*, pages 30–43, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[5] Lenka Turoňová, Lukáš Holík, Ondřej Lengál, Olli Saarikivi, Margus Veanes, and Tomáš Vojnar. Regex matching with counting-set automata. *Proc. ACM Program. Lang.*, 4(OOPSLA), nov 2020.

[6] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. When simulation meets antichains. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 158–174, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[7] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, aug 1986.

[8] Edmund M Clarke, Kenneth L McMillan, Xudong Zhao, Masahiro Fujita, and Jerry Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th international Design Automation Conference*, pages 54–60, 1993.

[9] Tom van Dijk. *Sylvan: multi-core decision diagrams*. PhD thesis, University of Twente, July 2016.

[10] Jeffrey Shallit. The frobenius problem and its generalizations. In Masami Ito and Masafumi Toyama, editors, *Developments in Language Theory*, pages 72–83, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Piotr Chrzastowski-Wachtel and Marek Raczunas. Liveness of weighted circuits and the diophantine problem of frobenius. In Zoltán Ésik, editor, *Fundamentals of Computation Theory*, pages 171–180, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.