Algebraic Reasoning Meets Automata in Solving Linear Integer Arithmetic

Peter Habermehl[0000−0002−7982−0946]2, Michal Hečko[1], Vojtěch Havlena[0000−0003−4375−7954]1, Lukáš Holík[0000−0001−6957−1651]1, and Ondřej Lengál[0000−0002−3038−5875]1

[1] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
[2] Université Paris Cité, IRIF, Paris, France

Abstract. We present a new angle on solving quantified linear integer arithmetic based on combining the automata-based approach, where numbers are understood as bitvectors, with ideas from (nowadays prevalent) algebraic approaches, which work directly with numbers. This combination is enabled by a fine-grained version of the duality between automata and arithmetic formulae. In particular, we employ a construction where states of automaton are obtained as derivatives of arithmetic formulae: then every state corresponds to a formula. Optimizations based on techniques and ideas transferred from the world of algebraic methods are used on thousands of automata states, which dramatically amplifies their effect. The merit of this combination of automata with algebraic methods is demonstrated by our prototype implementation being competitive to and even superior to state-of-the-art SMT solvers.

1 Introduction

Linear integer arithmetic (LIA), also known as *Presburger arithmetic*, is the first-order theory of integers with addition. Its applications include e.g. databases [57], program analysis [58], synthesis [56], and it is an essential component of every aspiring SMT solver. Many other types of constraints can either be reduced to LIA, or are decided using a tight collaboration of a solver for the theory and a LIA solver, e.g., in the theory of bitvectors [68], strings [19], or arrays [37]. Current SMT solvers are strong enough in solving large quantifier-free LIA formulae. Their ability to handle quantifiers is, however, problematic to the extent of being impractical. Even a tiny formula with two quantifier alternations can be a show stopper for them. Handling quantifiers is an area of lively research with numerous application possibilities waiting for a practical solution, e.g., software model checking [44], program synthesis [64], or theorem proving [46].

Among existing techniques for handling quantifiers, the complete approaches based on quantifier elimination [61,23] and automata [17,77,13] have been mostly deemed not scalable and abandoned in practice. Current SMT solvers use mainly incomplete techniques originating, e.g., from solving the theory of uninterpreted functions [63] and algebraic techniques, such as the *simplex* algorithm for quantifier-free formulae [25].

This work is the first step in leveraging a recent renaissance of practically competitive automata technology for solving LIA. This trend that has recently emerged in string constraint solving (e.g. [20,8,18,2,7]), processing regular expressions [24,21,72], reasoning about the SMT theory of bitvectors [51], or regex matching (e.g. [76,40,50,59]).

The new advances are rooted in paradigms such as usage of non-determinism and alternation, various flavours of symbolic representations, and combination with/or integration into SAT/SMT frameworks and with algebraic techniques.

We particularly show that the automata-based procedure provides unique opportunities to amplify certain algebraic optimizations that reason over the semantic of formulae. These optimizations then boost the inherent strong points of the automata-based approach to the extent that it is able to overcome modern SMT solvers. The core strong points of automata are orthogonal to those of algebraic methods, mainly due to treating numbers as strings of bits regardless of their numerical values. Automata can thus represent large sets of solutions succinctly and can use powerful techniques, such as minimization, that have no counterpart in the algebraic world. This makes automata more efficient than the algebraic approaches already in their basic form, implemented e.g. in [77,13], on some classes of problems such as the *Frobenius coin problem* [41]. In many practical cases, the automata construction, however, explodes. The explosion usually happens when constructing an intermediate automaton for a sub-formula, although the minimal automaton for the entire formula is almost always small. The plot in Fig. 1 shows that the gap between sizes of final and intermediate automata in our benchmark is always several orders of magnitude large, offering opportunities for optimizations. In this paper, we present a basic approach to breaching this gap by transferring techniques and ideas from the algebraic world to automata and using them to prune the vast state space.

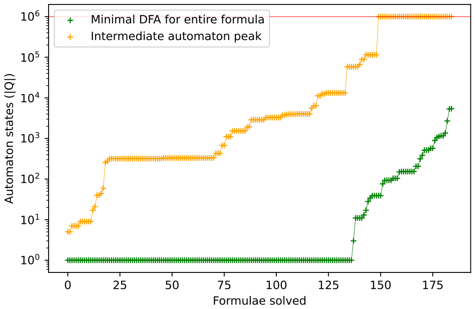


Fig. 1: Comparison of the peak intermediate automaton size and the size of the minimized DFA for the entire formula on the `SMT-COMP` benchmark (cf. Section 8).

To this end, we combine the classical inductive automata construction with constructing formula derivatives, similar to derivatives of regular expressions [15,3,72] or WS1S/WS*k*S formulae [75,32,43]. Our construction directly generates states of an automaton of a nested formula, without the need to construct intermediate automata for sub-formulae first. Although the derivative construction is not better than the inductive construction by itself, it gives an opportunity to optimize the state space *on the fly*, before it gets a chance to explode. The optimization itself is negotiated by the *fine-grained* version of the well-known *automaton-formula duality*. In the derivative construction, *every state* corresponds to a LIA formula. Applying equivalence-preserving formula rewriting on state formulae has the effect of merging or pruning states, similar to what DFA minimization could achieve after the entire automaton were constructed.

Our equivalence-preserving rewriting uses known algebraic techniques or ideas originating from them. First, we use basic formula simplification techniques, such as propagating true or false values or antiprenexing. Despite being simple, these simplifications have a large impact on performance. Second, we use *disjunction pruning*, which replaces $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_k$ by $\varphi_2 \vee \cdots \vee \varphi_k$ if φ_1 is entailed by the rest of the formula (this is close to the state pruning techniques used in [38,32,28]). We also adopt the principle of *quantifier instantiation* [26,65,36], where we detect cases when a quantified variable can be substituted by one or several values, or when a linear congruence can be simplified to a linear equation. We particularly use ideas from Cooper's quantifier elimination [23], where a quantifier is expanded into a disjunction over a finite number of values, and from Omega test [62], where a variable with a one-side unbounded range is substituted by the least restrictive value.

It is noteworthy that in the purely algebraic setting, the same techniques could only be applied once on the input formula, with a negligible effect. In the automata-based procedure, their power is amplified since they are used on thousands of derivative states generated deep within automata after reading several bits of the solution.

Our prototype implementation is competitive with the best SMT solvers on benchmarks from SMT-COMP, and, importantly, it is superior on quantifier-intensive instances. We believe that more connections along the outlined direction, based on the fine-grained duality between automata and formulae, can be found, and that the work in this paper is the first step in bridging the worlds of automata and algebraic approaches. Many challenges in incorporating automata-based LIA reasoning into SMT solvers still await but, we believe, can be tackled, as witnessed e.g. within the recent successes of the integration of automata-based string solvers [19,7,18].

2 Preliminaries

We use $\mathbb{Z}$ to denote the set of *integers*, $\mathbb{Z}^+$ to denote the set of *positive integers*, and $\mathbb{B}$ to denote the set of *binary digits* $\{0, 1\}$. For $x, y \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$, we use $x \equiv_m y$ to denote that x is congruent with y modulo m, i.e., there exists $z \in \mathbb{Z}$ s.t. $z \cdot m + x = y$; and $x|y$ to denote that there exists $z' \in \mathbb{Z}$ s.t. $y = z' \cdot x$. Furthermore, we use $[x]_m$ to denote the unique integer s.t. $0 \leq [x]_m < m$ and $x \equiv_m [x]_m$. The following notation will be used for intervals of integers: for $a, b \in \mathbb{Z}$, the set $\{x \in \mathbb{Z} \mid a \leq x \leq b\}$ is denoted as $[a, b]$, the set $\{x \in \mathbb{Z} \mid a \leq x\}$ is denoted as $[a, +\infty)$, and the set $\{x \in \mathbb{Z} \mid x \leq b\}$ is denoted as $(-\infty, b]$. The *greatest common divisor* of $a, b \in \mathbb{Z}$, denoted as $\gcd(a, b)$, is the largest integer such that $\gcd(a, b)|a$ and $\gcd(a, b)|b$ (note that $\gcd(a, 0) = |a|$); if $\gcd(a, b) = 1$, we say that a and b are *coprime*. For a real number y, $\lfloor y \rfloor$ denotes the *floor* of y, i.e., the integer $\max\{z \in \mathbb{Z} \mid z \leq y\}$, and $\lceil y \rceil$ denotes the *ceiling* of y, i.e., the integer $\min\{z \in \mathbb{Z} \mid z \geq y\}$.

An *alphabet* Σ is a finite non-empty set of *symbols* and a *word* $w = a_1 \ldots a_n$ of length n over Σ is a finite sequence of symbols from Σ. If $n = 0$, we call w the *empty word* and denote it ϵ. Σ^+ is the set of all non-empty words over Σ and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$.

Finite automata. In order to simplify constructions in the paper, we use a variation of finite automata with accepting transitions instead of states. A (*final transition acceptance-based) nondeterministic finite automaton* (FA) is a five-tuple $\mathcal{A} = (Q, \Sigma, \delta, I, Acc)$ where Q is a finite set of *states*, Σ is an *alphabet*, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, $I \subseteq Q$ is a set of *initial states*, and $Acc\colon \delta \to \{true, false\}$ is a transition-based acceptance condition. We often use $q \xrightarrow{a} p$ to denote that $(q, a, p) \in \delta$. A *run* of $\mathcal{A}$ over a word $w = a_1 \ldots a_n$ is a sequence of states $\rho = q_0 q_1 \ldots q_n \in Q^{n+1}$ such that for all $1 \leq i \leq n$ it holds that $q_{i-1} \xrightarrow{a_i} q_i$ and $q_0 \in I$. The run ρ is *accepting* if $n \geq 1$ and $Acc(q_{n-1} \xrightarrow{a_n} q_n)$

(i.e., if the last transition in the run is accepting)[3]. The language of $\mathcal{A}$, denoted as $\mathcal{L}(\mathcal{A})$, is defined as $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid$ there is an accepting run of $\mathcal{A}$ on $w\}$. We further use $\mathcal{L}_{\mathcal{A}}(q)$ to denote the language of the FA obtained from $\mathcal{A}$ by setting its set of initial states to $\{q\}$ (if the context is clear, we use just $\mathcal{L}(q)$).

$\mathcal{A}$ is *deterministic* (a DFA) if $|I| \leq 1$ and for all states $q \in Q$ and symbols $a \in \Sigma$, it holds that if $q \xrightarrow{a} p$ and $q \xrightarrow{a} r$, then $p = r$. On the other hand, $\mathcal{A}$ is *complete* if $|I| \geq 1$ and for all states $q \in Q$ and symbols $a \in \Sigma$, there is at least one state $p \in Q$ such that $q \xrightarrow{a} p$. For a deterministic and complete $\mathcal{A}$, we abuse notation and treat δ as a function $\delta\colon Q \times \Sigma \to Q$. A DFA $\mathcal{A}$ is *minimal* if $\forall q \in Q\colon \mathcal{L}(q) \neq \emptyset \wedge \forall p \in Q\colon p \neq q \Rightarrow \mathcal{L}(q) \neq \mathcal{L}(p)$. Hopcroft's [48] and Brzozowski's [14] algorithms for obtaining a minimal DFA can be modified for our definition of FAs .

Linear integer arithmetic. Let $\mathbb{X} = \{x_1, \ldots, x_n\}$ be a (finite) set of integer variables. We will use $\vec{x}$ to denote the vector $(x_1, \ldots, x_n)$. Sometimes, we will treat $\vec{x}$ as a set, e.g., $y \in \vec{x}$ denotes $y \in \{x_1, \ldots, x_n\}$. A *linear integer arithmetic* (LIA) formula φ over $\mathbb{X}$ is obtained using the following grammar:

$$\begin{aligned} \varphi_{atom} &::= \quad \vec{a} \cdot \vec{x} = c \mid \vec{a} \cdot \vec{x} \leq c \mid \vec{a} \cdot \vec{x} \equiv_m c \mid \bot \\ \varphi &::= \quad \varphi_{atom} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists y(\varphi) \end{aligned}$$

where $\vec{a}$ 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 $y \in \mathbb{X}$ (one can derive the other connectives $\top, \rightarrow, \leftrightarrow, \forall, \ldots$ in the standard way)[4]. Free variables of φ are denoted as $fv(\varphi)$. Given a formula φ, we say that an assignment $\nu\colon \mathbb{X} \to \mathbb{Z}$ is a *model* of φ, denoted as $\nu \models \varphi$, if ν satisfies φ in the standard way. Note that we use the same symbols $=, \leq, \equiv_m, \neg, \wedge, \vee, \exists, \ldots$ in the syntactical language (where they are not to be interpreted, with the exception of evaluation of constant expressions) of the logic as well as in the meta-language. In order to avoid ambiguity, we use the style φ for a syntactic formula. W.l.o.g. we assume that variables in φ are unique, i.e., there is no overlap between quantified variables and also between free and quantified variables.

In our decision procedure we represent integers as non-empty sequences of binary digits $a_0 \ldots a_n \in \mathbb{B}^+$ using the two's complement with the *least-significant bit first* (LSBF) encoding (i.e., the right-most bit denotes the *sign*). Formally, the *decoding* of a binary word represents the integer

$$\langle a_0 \ldots a_n \rangle = \sum_{i=0}^{n-1} a_i \cdot 2^i - a_n \cdot 2^n. \tag{1}$$

For instance, $\mathsf{dec}(0101) = -6$ and $\mathsf{dec}(010) = 2$. Note that any integer has infinitely many representations in this encoding: the shortest one and others obtained by repeating

[3] Note that our FAs cannot accept the empty word ϵ, which corresponds in our use to the fact that in the two's complement encoding of integers, one needs at least one bit (the sign bit) to represent a number, see further.

[4] Although the modulo constraint $\vec{a} \cdot \vec{x} \equiv_m c$ could be safely removed without affecting the expressivity of the input language, keeping it allows a more efficient automata construction and application of certain heuristics (cf. Section 7.1).

$$
\begin{aligned}
\mathit{Post}(\vec{a}\cdot\vec{x} \leq c\,,\sigma) &\overset{\text{def}}{=} \vec{a}\cdot\vec{x} \leq \lfloor \tfrac{1}{2}\kappa \rfloor \qquad\qquad \text{for } \kappa \overset{\text{def}}{=} c - \vec{a}\cdot\sigma \\
\mathit{Post}(\vec{a}\cdot\vec{x} = c\,,\sigma) &\overset{\text{def}}{=} \begin{cases} \vec{a}\cdot\vec{x} = \tfrac{1}{2}\kappa & \text{if } 2|\kappa \\ \bot & \text{otherwise} \end{cases} \\
\mathit{Post}(\vec{a}\cdot\vec{x} \equiv_{2m} c\,,\sigma) &\overset{\text{def}}{=} \begin{cases} \vec{a}\cdot\vec{x} \equiv_{m} \left[\tfrac{1}{2}\kappa\right]_{m} & \text{if } 2|\kappa \\ \bot & \text{otherwise} \end{cases} \\
\mathit{Post}(\vec{a}\cdot\vec{x} \equiv_{2m+1} c\,,\sigma) &\overset{\text{def}}{=} \begin{cases} \vec{a}\cdot\vec{x} \equiv_{2m+1} \left[\tfrac{1}{2}\kappa\right]_{2m+1} & \text{if } 2|\kappa \\ \vec{a}\cdot\vec{x} \equiv_{2m+1} \left[\tfrac{1}{2}(\kappa + 2m + 1)\right]_{2m+1} & \text{otherwise} \end{cases} \\
\mathit{Post}(\bot\,,\sigma) &\overset{\text{def}}{=} \bot
\end{aligned}
$$

Fig. 2: Definition of the transition function *Post* for atomic formulae. Note that the right-hand sides contain constant expressions, so they will be evaluated.

the sign bit any number of times. In this paper, we work with the so-called *binary assignments*. A binary assignment is an assignment $\nu\colon \mathbb{X} \to \mathbb{B}^{+}$ s.t. for each $x_1, x_2 \in \mathbb{X}$ the lengths of the words assigned to x_1 and x_2 match, i.e., $|\nu(x_1)| = |\nu(x_2)|$. We overload the decoding operator $\langle\cdot\rangle$ to binary assignments such that $\langle\nu\rangle\colon \mathbb{X} \to \mathbb{Z}$ is defined as $\langle\nu\rangle = \{x \mapsto \langle y\rangle \mid \nu(x) = y\}$. A *binary model* of a formula φ is a binary assignment ν such that $\langle\nu\rangle \models \varphi$. We denote the set of all binary models of a LIA formula φ as $[\![\varphi]\!]$ and we write $\varphi_1 \Rightarrow \varphi_2$ to denote $[\![\varphi_1]\!] \subseteq [\![\varphi_2]\!]$ and $\varphi_1 \Leftrightarrow \varphi_2$ to denote $[\![\varphi_1]\!] = [\![\varphi_2]\!]$.

3 Classical Automata-Based Decision Procedure for LIA

The following *classical decision procedure* is due to Boudet and Comon [13] (based on the ideas of [16]) with an extension to modulo constraints by Durand-Gasselin and Habermehl [29]. Given a set of variables $\mathbb{X}$, a symbol σ is a mapping $\sigma\colon \mathbb{X} \to \mathbb{B}$ and $\Sigma_{\mathbb{X}}$ denotes the set of all symbols over $\mathbb{X}$. For a symbol $\sigma \in \Sigma_{\mathbb{X}}$ and a variable $x \in \mathbb{X}$ we define the *projection* $\pi_x(\sigma) = \{\sigma' \in \Sigma_{\mathbb{X}} \mid \sigma'_{|\mathbb{X}\setminus\{x\}} = \sigma_{|\mathbb{X}\setminus\{x\}}\}$ where $\sigma_{|\mathbb{X}\setminus\{x\}}$ is the restriction of the function σ to the domain $\mathbb{X} \setminus \{x\}$.

For a LIA formula φ, the classical automata-based decision procedure builds an FA $\mathcal{A}_{\varphi}$ encoding all binary models of φ. We use a modification which uses automata with accepting edges instead of states. It allows to construct deterministic automata for atomic formulae, later in Section 4 also for complex formulae, and to eliminate an artificial final state present in the original construction that does not correspond to any arithmetic formula. The construction proceeds inductively as follows:

Base case. First, an FA $\mathcal{A}_{\varphi_{atom}}$ is constructed for each atomic formula φ_{atom} in φ. The states of $\mathcal{A}_{\varphi_{atom}}$ are LIA formulae with φ_{atom} being the (only) initial state. $\mathcal{A}_{\varphi_{atom}}$'s structure is given by the transition function *Post*, implemented via a derivative $\mathit{Post}(\varphi_{atom}\,,\sigma)$ of φ_{atom} w.r.t. symbols $\sigma \in \Sigma_{\mathbb{X}}$ as given in Fig. 2 (an example will follow).

Intuitively, for $\mathit{Post}(\vec{a}\cdot\vec{x} = c\,,\sigma)$, the next state after reading σ is given by taking the least significant bits (LSBs) of all variables ($\vec{x}$) after being multiplied with the respective coefficients ($\vec{a}$) and subtracting this value from c. If the parity of the result is odd, we can terminate ($\vec{a} \cdot \vec{x}$ and c have a different LSB, so they cannot match), otherwise we can remove the LSB of the result, set it as a new c, and continue. One can imagine this process as performing a long addition of several binary numbers at once with c being the result (the subtraction from c can be seen as working with *carry*). The intuition for a formula $\vec{a} \cdot \vec{x} \leq c$ is similar. On the other hand, for a formula $\vec{a} \cdot \vec{x} \equiv_{2m} c$, i.e., a congruence with an even modulus, if the parity of the left-hand side ($\vec{a} \cdot \vec{x}$) and the right-hand side (c) does not match (in other words, $c - \vec{a} \cdot \vec{x}$ is odd), because the modulus is even we can terminate. Otherwise, we remove the LSB of the modulus (i.e., divide it by two). Lastly, let us mention the second case for the rule for a formula of the form $\vec{a} \cdot \vec{x} \equiv_{2m+1} c$. Here, since κ is odd, we cannot divide it by two; however, adding the modulus $(2m + 1)$ to κ yields an even value equivalent to κ.

The states of $\mathcal{A}_{\varphi_{atom}}$ are then all reachable formulae obtained from the application of *Post* from the initial state, with the reachability from a set of formulae S using symbols from Γ given using the least fixpoint operator μ as follows:

$$Reach(S, \Gamma) = \mu Z \colon S \cup \{Post(\psi, a) \mid \psi \in Z, a \in \Gamma\} \tag{2}$$

Lemma 1. *$Reach(\{\varphi_{atom}\}, \Sigma_{\mathbb{X}})$ is finite for an atomic formula φ_{atom}.*

Proof. The cases for linear equations and inequations follow from [13, Proposition 1] and [13, Proposition 3] respectively. For moduli, the lemma follows from the fact that in the definition of *Post*, the right-hand side of a modulo is an integer from $[0, m - 1]$. □

$$\begin{aligned}
Fin(\vec{a} \cdot \vec{x} \leq c, \sigma) &\overset{\text{def}}{\Leftrightarrow} c + \vec{a} \cdot \sigma \geq 0 \\
Fin(\vec{a} \cdot \vec{x} = c, \sigma) &\overset{\text{def}}{\Leftrightarrow} c + \vec{a} \cdot \sigma = 0 \\
Fin(\vec{a} \cdot \vec{x} \equiv_m c, \sigma) &\overset{\text{def}}{\Leftrightarrow} c + \vec{a} \cdot \sigma \equiv_m 0 \\
Fin(\bot, \sigma) &\overset{\text{def}}{\Leftrightarrow} \mathit{false}
\end{aligned}$$

Fig. 3: Acceptance for atomic formulae.

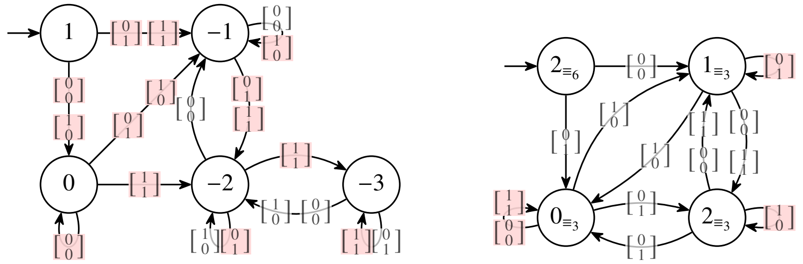
Post is deterministic, so it suffices to define the acceptance condition for the derivatives only for each state and symbol, as given in Fig. 3. E.g., a transition from $2x_1 - 7x_2 = 5$ over $\sigma = \left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$ is accepting; the intuition is similar as for *Post* with the difference that the last bit is the sign bit (cf. Eq. (1)), so it is treated in the opposite way to other bits (therefore, there is the "+" sign on the right-hand sides of the definitions rather than the "−" sign as in Fig. 2). If we substitute into the example, we obtain $2 \cdot (-1) - 7 \cdot (-1) = -2 + 7 = 5$. The acceptance condition *Acc* is then defined as $Acc(\varphi_1 \xrightarrow{\sigma} Post(\varphi_1, \sigma)) \overset{\text{def}}{=} Fin(\varphi_1, \sigma)$ and $\mathcal{A}_{\varphi_{atom}}$ is defined as the FA

$$\mathcal{A}_{\varphi_{atom}} = (Reach(\{\varphi_{atom}\}, \Sigma_{\mathbb{X}}), \Sigma_{\mathbb{X}}, Post, \{\varphi_{atom}\}, Acc). \tag{3}$$

Note that if an FA accepts a word w, it also accepts all words obtained by appending any number of copies of the most significant bit (the sign) to w.

Example 1. Fig. 4 gives examples of FAs for $x + 2y \leq 1$ and $x + 2y \equiv_6 2$. □

Inductive case. The inductive cases for Boolean connectives are defined in the standard way: conjunction of two formulae is implemented by taking the intersection of the two corresponding FAs, disjunction by taking their union, and negation is implemented by taking the complement (which may involve determinization via the subset construction). Formally, let $\mathcal{A}_{\varphi_i} = (Q_{\varphi_i}, \Sigma_{\mathbb{X}}, \delta_{\varphi_i}, I_{\varphi_i}, Acc_{\varphi_i})$ for $i \in \{1, 2\}$ with $Q_{\varphi_1} \cap Q_{\varphi_2} = \emptyset$ be complete FAs. Then,



(a) The FA for $x + 2y \leq 1$. A state $x + 2y \leq c$ is represented by “c”.

(b) The FA for $x + 2y \equiv_6 2$. A state $x + 2y \equiv_m c$ is represented by “$c_{\equiv_m}$”.

Fig. 4: Examples of FAs for atomic formulae. The notation for symbols is $[\begin{smallmatrix}x\\y\end{smallmatrix}]$; red background denotes accepting transitions.

- $\mathcal{A}_{\varphi_1\wedge\varphi_2} = (Q_{\varphi_1} \times Q_{\varphi_2}, \Sigma_{\mathbb{X}}, \delta_{\varphi_1\wedge\varphi_2}, I_{\varphi_1} \times I_{\varphi_2}, Acc_{\varphi_1\wedge\varphi_2})$ where
  - $\delta_{\varphi_1\wedge\varphi_2} = \{(q_1, q_2) \xrightarrow{\sigma} (p_1, p_2) \mid q_1 \xrightarrow{\sigma} p_1 \in \delta_{\varphi_1}, q_2 \xrightarrow{\sigma} p_2 \in \delta_{\varphi_2}\}$ and
  - $Acc_{\varphi_1\wedge\varphi_2}((q_1, q_2) \xrightarrow{\sigma} (p_1, p_2)) \overset{\text{def}}{\Leftrightarrow} Acc_{\varphi_1}(q_1 \xrightarrow{\sigma} p_1) \wedge Acc_{\varphi_2}(q_2 \xrightarrow{\sigma} p_2)$.
- $\mathcal{A}_{\varphi_1\vee\varphi_2} = (Q_{\varphi_1} \times Q_{\varphi_2}, \Sigma_{\mathbb{X}}, \delta_{\varphi_1\vee\varphi_2}, I_{\varphi_1} \times I_{\varphi_2}, Acc_{\varphi_1\vee\varphi_2})$ where
  - $\delta_{\varphi_1\vee\varphi_2} = \{(q_1, q_2) \xrightarrow{\sigma} (p_1, p_2) \mid q_1 \xrightarrow{\sigma} p_1 \in \delta_{\varphi_1}, q_2 \xrightarrow{\sigma} p_2 \in \delta_{\varphi_2}\}$ and
  - $Acc_{\varphi_1\vee\varphi_2}((q_1, q_2) \xrightarrow{\sigma} (p_1, p_2)) \overset{\text{def}}{\Leftrightarrow} Acc_{\varphi_1}(q_1 \xrightarrow{\sigma} p_1) \vee Acc_{\varphi_2}(q_2 \xrightarrow{\sigma} p_2)$.
- $\mathcal{A}_{\neg\varphi_1} = (2^{Q_{\varphi_1}}, \Sigma_{\mathbb{X}}, \delta_{\neg\varphi_1}, \{I_{\varphi_1}\}, Acc_{\neg\varphi_1})$ where
  - $\delta_{\neg\varphi_1} = \{S \xrightarrow{\sigma} T \mid T = \{p \in Q_{\varphi_1} \mid \exists q \in S : q \xrightarrow{\sigma} p \in \delta_{\varphi_1}\}$ and
  - $Acc_{\neg\varphi_1}(S \xrightarrow{\sigma} T) \overset{\text{def}}{\Leftrightarrow} \forall q \in S\ \forall p \in T : \neg Acc_{\varphi_1}(q \xrightarrow{\sigma} p)$.

Existential quantification is more complicated. Given a formula $\exists x(\varphi)$ and the FA $\mathcal{A}_\varphi = (Q_\varphi, \Sigma_{\mathbb{X}}, \delta_\varphi, I_\varphi, Acc_\varphi)$, a word w should be accepted by $\mathcal{A}_{\exists x(\varphi)}$ iff there is a word w' accepted by $\mathcal{A}_\varphi$ s.t. w and w' are the same on all tracks except the track for x. One can perform projection of x out of $\mathcal{A}_\varphi$, i.e., remove the x track from all its transitions. This is, however, insufficient. For instance, consider the model $\{x \mapsto 7, y \mapsto -4\}$, encoded into the (shortest) word $[\begin{smallmatrix}1\\0\end{smallmatrix}][\begin{smallmatrix}1\\0\end{smallmatrix}][\begin{smallmatrix}1\\1\end{smallmatrix}][\begin{smallmatrix}0\\1\end{smallmatrix}]$ (we use the notation $[\begin{smallmatrix}x\\y\end{smallmatrix}]$). When we remove the x-track from the word, we obtain [0][0][1][1], which encodes the assignment $\{y \mapsto -4\}$. It is, however, not the shortest encoding of the assignment; the shortest encoding is [0][0][1]. Therefore, we further need to modify the FA obtained after projection to also accept words that would be accepted if their sign bit were arbitrarily extended, which we do by reachability analysis on the FA. Formally, $\mathcal{A}_{\exists x(\varphi)} = (Q_\varphi, \Sigma_{\mathbb{X}}, \delta_{\exists x(\varphi)}, I_\varphi, Acc_{\exists x(\varphi)})$ where

- $\delta_{\exists x(\varphi)} = \{q \xrightarrow{\sigma'} p \mid \exists q \xrightarrow{\sigma} p \in \delta_\varphi : \sigma' \in \pi_x(\sigma)\}$ and
- $Acc_{\exists x(\varphi)}(q \xrightarrow{\sigma} p) \overset{\text{def}}{\Leftrightarrow} \bigvee_{\sigma'\in\pi_x(\sigma)} Acc_\varphi(q \xrightarrow{\sigma'} p) \vee \exists r, s \in Reach(\{p\}, \pi_x(\sigma))\colon \bigvee_{\sigma'\in\pi_x(\sigma)} Acc_\varphi(r \xrightarrow{\sigma'} s)$.

After defining the base and inductive cases for constructing the FA $\mathcal{A}_\varphi$, we can establish the connection between its language and the models of φ. For a word $w = a_1 \ldots a_n \in \Sigma_{\mathbb{X}}$ and a variable $x \in \mathbb{X}$, we define $w_x = a_1(x) \ldots a_n(x)$, i.e., w_x extracts the binary number assigned to variable x in w. For a binary assignment ν of a LIA formula φ, we define its language as $\mathcal{L}(\nu) = \{w \in \Sigma_{\mathbb{X}}^* \mid \forall x \in \mathbb{X}\colon w_x = \nu(x)\}$. We lift the language to sets of binary assignments as usual.

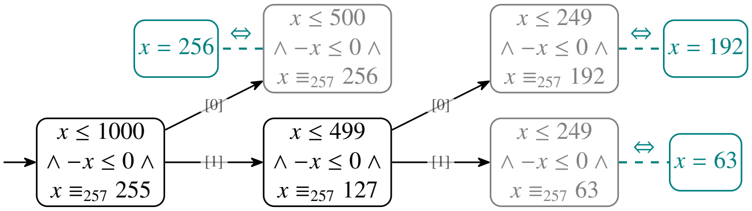

$x = 256$
$\Leftrightarrow$
$x \leq 500 \wedge -x \leq 0 \wedge x \equiv_{257} 256$
$x \leq 249 \wedge -x \leq 0 \wedge x \equiv_{257} 192$
$\Leftrightarrow$
$x = 192$
[0]
[0]
$x \leq 1000 \wedge -x \leq 0 \wedge x \equiv_{257} 255$
[1]
$x \leq 499 \wedge -x \leq 0 \wedge x \equiv_{257} 127$
[1]
$x \leq 249 \wedge -x \leq 0 \wedge x \equiv_{257} 63$
$\Leftrightarrow$
$x = 63$

Fig. 6: Example of rewriting formulae in the FA for $0 \leq x \leq 1000 \wedge x \equiv_{257} 255$.

Theorem 1. *Let φ be a LIA formula. Then $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}([\![\varphi]\!])$.*

Proof. Follows from [13, Lemma 5].

4 Derivative-based Construction for Nested Formulae

This section lays down the basics of our approach to interconnecting the automata with the algebraic approach for quantified LIA. We aim at using methods and ideas from the algebraic approach to circumvent the large intermediate automata constructed along the way before obtaining the small DFAs (cf. Fig. 1). To do that, we need a variation of the automata-based decision procedure that exposes the states of the target automaton without the need of generating the complete state space of the intermediate automata first. To achieve this, we generalize the post-image function *Post* (and the acceptance condition *Fin*) from Section 3 to general non-atomic formulae using an approach similar to that of [32,43,74], which introduced derivatives of WS1S/WS*k*S formulae. Computing formula derivatives produces automata states that are at the same time LIA formulae, and can be manipulated as such using algebraic methods and reasoning about their integer semantics. We will then use basic Boolean simplification, antiprenexing, and also ideas from Cooper's algorithm and Omega test [23,62] to prune and simplify the state-formulae. These particular techniques will be discussed in Sections 5 to 7.

$$\begin{aligned}
Post(\varphi_1 \wedge \varphi_2, \sigma) &\overset{\text{def}}{=} Post(\varphi_1, \sigma) \wedge Post(\varphi_2, \sigma)\\
Post(\varphi_1 \vee \varphi_2, \sigma) &\overset{\text{def}}{=} Post(\varphi_1, \sigma) \vee Post(\varphi_2, \sigma)\\
Post(\neg\varphi, \sigma) &\overset{\text{def}}{=} \neg Post(\varphi, \sigma)\\
Post(\exists x(\varphi), \sigma) &\overset{\text{def}}{=} \exists x \left(\textstyle\bigvee_{\sigma' \in \pi_x(\sigma)} Post(\varphi, \sigma') \right)
\end{aligned}$$

$$\begin{aligned}
Fin(\neg\varphi, \sigma) &\overset{\text{def}}{\Leftrightarrow} \neg Fin(\varphi, \sigma)\\
Fin(\varphi_1 \wedge \varphi_2, \sigma) &\overset{\text{def}}{\Leftrightarrow} Fin(\varphi_1, \sigma) \wedge Fin(\varphi_2, \sigma)\\
Fin(\varphi_1 \vee \varphi_2, \sigma) &\overset{\text{def}}{\Leftrightarrow} Fin(\varphi_1, \sigma) \vee Fin(\varphi_2, \sigma)\\
Fin(\exists x(\varphi), \sigma) &\overset{\text{def}}{\Leftrightarrow} \exists \psi \in Reach(\{\varphi\}, \pi_x(\sigma)) :\\
&\qquad \textstyle\bigvee_{\sigma' \in \pi_x(\sigma)} Fin(\psi, \sigma')
\end{aligned}$$

Fig. 5: *Post* and *Fin* for non-atomic formulae.

Example 2. In Fig. 6, we show an intuitive example of rewriting state formulae when constructing the FA for $0 \leq x \leq 1000 \wedge x \equiv_{257} 255$. After reading the first symbol [0], the obtained state formula $0 \leq x \leq 500 \wedge x \equiv_{257} 256$ is satisfied only by one x, which is 256. We can therefore rewrite the formula into an equivalent formula $x = 256$. Similar rewriting can be applied to the state obtained after reading [1][0] and [1][1]. The rest of the automaton constructed from the rewritten states $x = 256$, $x = 192$, and $x = 63$

is then of a logarithmic size (each state in the rest will have only one successor based on the binary encoding of 256, 192, or 63 respectively, while if we did not perform the rewriting, the states would have two successors and the size would be linear). □

In Fig. 5, we extend the derivative post-image function *Post* and the acceptance condition *Fin* (cf. Figs. 2 and 3) to non-atomic formulae. The derivatives mimic the automata constructions in Section 3, with the exception that at every step, the derivative (and therefore also the state in the constructed FA) is a LIA formula and can be treated as such. One notable exception is $Post(\exists x(\varphi), \sigma)$, which, since the *Post* function is deterministic, in addition to the projection, also mimics determinisation. One can see the obtained disjunction-structure as a set of states from the standard subset construction in automata. Correctness of the construction is stated in the following.

Lemma 2. *Let φ be a LIA formula and let $\mathcal{A}_\varphi$ be the FA constructed by the procedure in this section or any combination of it and the classical one. Then $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\llbracket \varphi \rrbracket)$.*

Proof. Follows from preservation of languages of the states/formulae. □

Without optimizations, the derivative-based construction would generate a larger FA than the one obtained from the classical construction, which can perform minimization of the intermediate automata. The derivative-based construction cannot minimize the intermediate automata since they are not available; they are in a sense constructed on the fly within the construction of the automaton for the entire formula. Our algebraic optimizations mimic some effect of the minimization on the fly, while constructing the automaton, by simplifying the state formulae and detecting entailment between them.

In principle, when we construct a state q of $\mathcal{A}_\psi$ as a result of *Post*, we could test whether some state p was already constructed such that $\varphi_q \Leftrightarrow \varphi_p$ and, if so, we could merge p and q (drop q and redirect the edges to p). This would guarantee us to directly obtain the minimal DFA for ψ (no two states would be language-equivalent).

Solving the LIA equivalence queries precisely is, however, as hard as solving the original problem. Even when we restrict ourselves to quantifier-free formulae, the equivalence problem is **co-NP**-complete. Our algebraic optimizations are thus a cheaper and more practical alternative capable of merging at least some equivalent states. We discuss the optimizations in detail in Sections 5 to 7 and also give a comprehensive example of their effect in Appendix A.

5 Simple Rewriting Rules

The simplest rewriting rules are just common simplifications generally applicable in predicate logic. Despite their simplicity, they are quite powerful, since their use enables to apply the other optimizations (Sections 6 and 7) more often.

1. We apply the propositional laws of *identity* ($\varphi \vee \bot = \varphi$ and $\varphi \wedge \top = \varphi$) and *annihilation* ($\varphi \wedge \bot = \bot$ and $\varphi \vee \top = \top$) to simplify the formulae.
2. We use *antiprenexing* [42,30] (i.e., pushing quantifiers as deep as possible using inverses of prenexing rules [66, Chapter 5]). This is helpful, e.g., after a range-based

quantifier instantiation (cf. Section 7.2), which yields a disjunction. Since our formula analysis framework (Section 7) only works over conjunctions below existential quantifiers, we need to first push existential quantifiers inside the disjunctions to allow further applications of the heuristics.

3. Since negation is implemented as automaton complementation, we apply *De Morgan's laws* ($\neg(\varphi_1 \wedge \varphi_2) \Leftrightarrow (\neg\varphi_1) \vee (\neg\varphi_2)$ and $\neg(\varphi_1 \vee \varphi_2) \Leftrightarrow (\neg\varphi_1) \wedge (\neg\varphi_2)$) to push negation as deep as possible. The motivation is that small subformulae are likely to have small corresponding automata. As complementation requires the underlying automaton to be deterministic, complementing smaller automata helps to mitigate the exponential blow-up of determinization.

Moreover, we also employ the following simplifications valid for LIA:

4. We apply simple reasoning based on variable bounds to simplify the formula, e.g., $x \geq 0 \wedge x \leq 10 \wedge x \neq 0 \Leftrightarrow x \geq 1 \wedge x \leq 10$, and to prune away some parts of the formula, e.g., $x \geq 3 \wedge (\varphi \vee (x = 0 \wedge \psi)) \Leftrightarrow x \geq 3 \wedge \varphi$.
5. We employ rewriting rules aimed at accelerating the automata construction by minimizing the number of variables used in a formula, and, thus, avoiding constructing complicated transition relations, e.g., $\exists x_1, x_2(ay + b_1x_1 + b_2x_2 \equiv_K 0) \Leftrightarrow \exists x(ay + bx \equiv_K 0)$ where $b = \gcd(b_1, b_2)$, or $\exists x(ay + bx = 0) \Leftrightarrow ay \equiv_{|b|} 0$.
6. We detect conflicts by identifying small isomorphic subformulae, i.e., subformulae that have the same abstract syntax tree, except for renaming of quantified variables, for example, $\exists x(x > 3 \wedge x + z \leq 10) \wedge \neg(\exists y(y > 3 \wedge y + z \leq 10) \Leftrightarrow \bot$. One can see this as a variant of DAGification used in Mona [54].

6 Disjunction Pruning

We prune disjunctions by removing disjunct implied by other disjuncts. That is, if it holds that $\varphi_2 \vee \cdots \vee \varphi_k \Rightarrow \varphi_1$, then $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_k$ can be replaced by just $\varphi_2 \vee \cdots \vee \varphi_k$. Testing the entailment precisely is hard, so we use a stronger but cheaper relation of *subsumpion*. Our subsumption is a preorder (a reflexive and transitive relation) $\preceq_{\mathsf{s}}$ between LIA formulae in Fig. 7[5]. When we encounter the said macrostate and establish $\varphi_1 \preceq_{\mathsf{s}} \varphi_2 \vee \cdots \vee \varphi_k$, we perform the rewriting. This optimization has effect mainly in formulae of the form $\exists x(\psi)$: their *Post* contains a disjunction of formulae of a similar structure.

$$\vec{a_1} \cdot \vec{x_1} \leq c_1 \preceq_{\mathsf{s}} \vec{a_2} \cdot \vec{x_2} \leq c_2 \overset{\text{def}}{\Leftrightarrow} \vec{a_1} = \vec{a_2} \wedge \vec{x_1} = \vec{x_2} \wedge c_1 \leq c_2$$
$$\textstyle\bigwedge_{i=1}^{n} \varphi_i \preceq_{\mathsf{s}} \bigwedge_{j=1}^{m} \psi_j \overset{\text{def}}{\Leftrightarrow} \forall 1 \leq j \leq m \, \exists 1 \leq i \leq n \colon \varphi_i \preceq_{\mathsf{s}} \psi_j$$
$$\textstyle\bigvee_{i=1}^{n} \varphi_i \preceq_{\mathsf{s}} \bigvee_{j=1}^{m} \psi_j \overset{\text{def}}{\Leftrightarrow} \forall 1 \leq i \leq n \, \exists 1 \leq j \leq m \colon \varphi_i \preceq_{\mathsf{s}} \psi_j$$
$$\neg\varphi \preceq_{\mathsf{s}} \neg\psi \overset{\text{def}}{\Leftrightarrow} \psi \preceq_{\mathsf{s}} \varphi$$
$$\exists x(\varphi) \preceq_{\mathsf{s}} \exists x(\psi) \overset{\text{def}}{\Leftrightarrow} \varphi \preceq_{\mathsf{s}} \psi$$

Fig. 7: Definition of the subsumption preorder $\preceq_{\mathsf{s}}$ (we omit cases implied by reflexivity).

[5] The subsumption is similar to the one used in efficient decision procedures for WS1S/WS*k*S [32,43] with two important differences: (i) it can look inside atomic formulae and use semantics of states and (ii) it does not depend on the initial structure of the initial formula. Both of these make the subsumption relation larger.

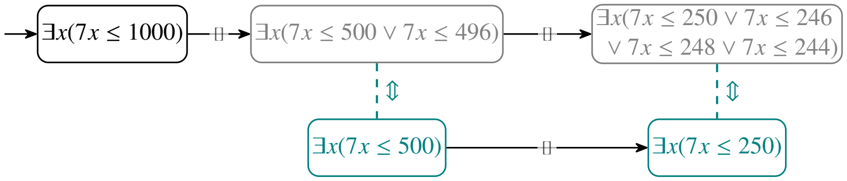

$\exists x(7x \leq 1000)$
[]
$\exists x(7x \leq 500 \vee 7x \leq 496)$
[]
$\exists x(7x \leq 250 \vee 7x \leq 246$
$\vee\, 7x \leq 248 \vee 7x \leq 244)$
$\Updownarrow$
$\Updownarrow$
$\exists x(7x \leq 500)$
[]
$\exists x(7x \leq 250)$

Fig. 8: Example of disjunction pruning in the FA for $\exists x(7x \leq 1000)$.

Lemma 3. *For LIA formulae φ_1 and φ_2, if $\varphi_1 \preceq_{\mathsf{s}} \varphi_2$, then $\varphi_1 \Rightarrow \varphi_2$.*

Example 3. In Fig. 8, we show an example of pruning disjunctions in the FA for the formula $\exists x(7x \leq 1000)$. □

7 Quantifier Instantiation

The next optimization is an instance of quantifier instantiation, a well known class of algebraic techniques. We gather information about the formulae with a focus on the way a particular variable, usually a quantified one, affects the models of the whole formula. If one can find "the best" value for such a variable (e.g., a value such that using it preserves all models of the formula), then the (quantified) variable can be substituted with a concrete value. For instance, let $\varphi = \exists y(x - y \leq 33 \wedge y \leq 12 \wedge y \equiv_7 2)$. The variable y is quantified so we can think about instantiating it (it will not occur in a model). The first atom $x - y \leq 33$ says that we want to pick y as large as possible (larger y's have higher chance to satisfy the inequation), but, on the other hand, the second atom $y \leq 12$ says that y can be at most 12. The last atom $y \equiv_7 2$ adds an additional constraint on y. Intuitively, we can see that the best value of y—i.e., the value that preserves all models of φ—would be 9, allowing to rewrite φ to $x \leq 42$.

To define the particular ways of gathering such kind of information in a uniform way, we introduce the following *formula analysis* framework that uses the function FA to extract information from formulae. Consider a *meet-semilattice* $(\mathbb{D}, \sqcap)$ where *undef* $\in \mathbb{D}$ is the bottom element. Let atom be a function that, given an atomic formula φ_{atom} and a variable $y \in \mathbb{X}$, outputs an element of $\mathbb{D}$ that represents the behavior of y in φ_{atom} (e.g., bounds on y). The function FA then aggregates the information from atoms into an information about the behavior of y in the whole formula using the meet operator $\sqcap$ recursively as follows:

$$\mathsf{FA}(\varphi_{atom}, y, \mathsf{atom}, \sqcap) = \mathsf{atom}(\varphi_{atom}, y) \tag{4}$$

$$\mathsf{FA}(\varphi_1 \wedge \varphi_2, y, \mathsf{atom}, \sqcap) = \mathsf{FA}(\varphi_1, y, \mathsf{atom}, \sqcap) \sqcap \mathsf{FA}(\varphi_2, y, \mathsf{atom}, \sqcap) \tag{5}$$

(By default, a missing case in the pattern matching evaluates to *undef*.) We note that the framework is defined only for conjunctions of formulae, which is the structure of subformulae that was usually causing troubles in our experiments (cf. Section 8).

The optimizations defined later are based on substituting certain variables in a formula with concrete values to obtain an equivalent (simpler) formula. For this, we extend standard substitution as follows. Let $\varphi(\vec{x}, y)$ be a formula with free variables $\vec{x} = (x_1, \ldots, x_n)$ and $y \notin \vec{x}$. For $k \in \mathbb{Z}$, substituting k for y in φ yields the formula $\varphi[y/k]$

obtained in the usual way (with all constant expressions being evaluated). For $k = \pm\infty$, the resulting formula is obtained for inequalities containing y as

$$(\vec{a} \cdot \vec{x} + a_y \cdot y \leq c)[y/k] = \top \quad \text{if } a_y \cdot k = -\infty \tag{6}$$

and is undefined for all other atomic formulae.

7.1 Quantifier Instantiation based on Formula Monotonicity

The first optimization based on quantifier instantiation uses the so-called *monotonicity* of formulae w.r.t. some variables (a similar technique is used in the Omega test [62]). Consider the following two formulae:

$$\varphi_1 = \exists y(\psi \land 3y - x \geq z) \quad \text{and} \quad \varphi_2 = \exists y(\psi \land 3y - x \geq z \land 5y \leq 42) \tag{7}$$

where ψ does not contain occurrences of y, and x, z are free variables in φ_1, φ_2. For φ_1, since y is existentially quantified, the inequation $3y - x \geq z$ can be always satisfied by picking an arbitrarily large value for y, so φ_1 can be simplified to just ψ. On the other hand, for φ_2, we cannot pick an arbitrarily large y because of the other inequation $5y \leq 42$. We can, however, observe, that $\lfloor \frac{42}{5} \rfloor = 8$ is the largest value that we can substitute for y to satisfy $5y \leq 42$. As a consequence, since the possible value of y in $3y - x \geq z$ is not bounded from above, we can substitute y by the value 8, i.e., φ_2 can be simplified to $\psi \land 24 - x \geq z$.

Formally, let $c \in \mathbb{Z} \cup \{+\infty\}$ and $y \in \mathbb{X}$. We say that a formula $\varphi(\vec{x}, y)$ is *c-best from below w.r.t. y* if (i) $[\![\varphi[y/y_1]]\!] \subseteq [\![\varphi[y/y_2]]\!]$ for all $y_1 \leq y_2 \leq c$ (for $c \in \mathbb{Z}$) or for all $y_1 \leq y_2$ (for $c = +\infty$) and (ii) $[\![\varphi[y/y']]\!] = \emptyset$ for all $y' > c$. Intuitively, substituting bigger values for y (up to c) in φ preserves all models obtained by substituting smaller values, so c can be seen as the most conservative limit of y (and for $c = +\infty$, this means that y does not have an upper bound, so it can be chosen arbitrarily large for concrete values of other variables). Similarly, $\varphi(\vec{x}, y)$ is called *c-best from above* (w.r.t. y) for $c \in \mathbb{Z} \cup \{-\infty\}$ if (i) $[\![\varphi[y/y_1]]\!] \subseteq [\![\varphi[y/y_2]]\!]$ for all $y_1 \geq y_2 \geq c$ (for $c \in \mathbb{Z}$) or for all $y_1 \geq y_2$ (for $c = -\infty$) and (ii) $[\![\varphi[y/y']]\!] = \emptyset$ for all $y' < c$. If a formula is c-best from below or above, we call it *c-monotone* (w.r.t. y).

Lemma 4. *Let $c \in \mathbb{Z} \cup \{\pm\infty\}$ and $\varphi(\vec{x}, y)$ be a formula c-monotone w.r.t. y such that $\varphi[y/c]$ is defined. Then the formula $\exists y(\varphi(\vec{x}, y))$ is equivalent to the formula $\varphi[y/c]$.*

Moreover, the following lemma utilizes formula monotonicity to provide a tool for simplification of formulae containing a modulo atom.

Lemma 5. *Let $c \in \mathbb{Z}$ and $\varphi(\vec{x}, y)$ be a formula c-monotone w.r.t. y for $c \in \mathbb{Z}$. Then, the formula $\exists y(\varphi(\vec{x}, y) \land y \equiv_m k)$ is equivalent to the formula $\varphi[y/c']$ where (i) $c' = \max\{\ell \in \mathbb{Z} \mid \ell \equiv_m k, \ell \leq c\}$ if φ is c-best from below, and (ii) $c' = \min\{\ell \in \mathbb{Z} \mid \ell \equiv_m k, \ell \geq c\}$ if φ is c-best from above.*

In general, it is, however, expensive to decide whether a formula is c-monotone and find the tight c. Therefore, we propose a cheap approximation working on the structure of LIA formulae, which uses the formula analysis function FA introduced above. First, we propose the partial function $\mathsf{blw}_{atom}(\varphi, y)$ (whose result is in $\mathbb{Z} \cup \{+\infty\}$) estimating the c for atomic formulae c-best from above w.r.t. y:

$$\mathsf{blw}_{atom}(a \cdot y \leq c, y) = \left\lfloor \tfrac{c}{a} \right\rfloor \qquad \text{if } a > 0 \tag{8}$$

$$\mathsf{blw}_{atom}(\vec{a} \cdot \vec{x} \leq c, x_i) = +\infty \qquad \text{if } a_i = 0 \vee \exists j \colon i \neq j \wedge a_j \neq 0 \wedge a_i < 0 \tag{9}$$

Intuitively, if y is in an inequation $a \cdot y \leq c$ without any other variable and $a > 0$, then y's value is bounded from above by $\left\lfloor \frac{c}{a} \right\rfloor$. On the other hand, if $y = x_i$ is in an inequation $\vec{a} \cdot \vec{x} \leq c$ where $\vec{a}$ has at least two nonzero coefficients and y's coefficient is negative, or y does not appear in the inequation at all, then y's value is not bounded (larger values of y make it easier to satisfy the inequation). The value for other cases is undefined.

Similarly, $\mathsf{abv}_{atom}(\varphi, y)$ (with the result in $\mathbb{Z} \cup \{-\infty\}$) estimates the c for atomic formulae c-best from above:

$$\mathsf{abv}_{atom}(a \cdot y \leq c, y) = \left\lfloor \tfrac{c}{a} \right\rfloor \qquad \text{if } a < 0 \tag{10}$$

$$\mathsf{abv}_{atom}(\vec{a} \cdot \vec{x} \leq c, x_i) = -\infty \qquad \text{if } a_i = 0 \vee \exists j \colon i \neq j \wedge a_j \neq 0 \wedge a_i > 0 \tag{11}$$

Based on blw_{atom} and abv_{atom} and using the FA framework, we define the functions blw and abv estimating the c for general formulae c-best from below and above as

$$\mathsf{blw}(\varphi, y) = \mathsf{FA}(\varphi, y, \mathsf{blw}_{atom}, \min), \quad \mathsf{abv}(\varphi, y) = \mathsf{FA}(\varphi, y, \mathsf{abv}_{atom}, \max). \tag{12}$$

For a formula $\psi = \exists y(\varphi(\vec{x}, y) \wedge y \equiv_m k)$, the simplification algorithm then determines whether φ is c-monotone for some c, which is done using the abv and blw functions. In particular, if $\mathsf{blw}(\varphi, y) = c$ for some $c \in \mathbb{Z} \cup \{\pm\infty\}$, we have that φ is c-best from below w.r.t. y (analogously for abv). Then, in the positive case and if $c \in \mathbb{Z}$, we apply Lemma 5 to simplify the formula ψ. If ψ is of the simple form $\exists y(\varphi(\vec{x}, y))$ where φ is c-monotone w.r.t. y, we can directly use Lemma 4 to simplify ψ to $\varphi[y/c]$.

Example 4. Consider the formula $\psi = \exists y(x - y \leq 1 \wedge y \leq -1 \wedge y \equiv_5 0)$. In order to simplify ψ, we first need to check if the formula $\varphi(x, y) = x - y \leq 1 \wedge y \leq -1$ is c-monotone. Using blw, we can deduce that $\mathsf{blw}(x - y \leq 1, y) = \infty$, $\mathsf{blw}(y \leq -1, y) = -1$, and hence $\mathsf{blw}(\varphi, y) = -1$ meaning that φ is (-1)-best from below w.r.t. y ($\mathsf{abv}(\varphi, y)$ is undefined). Lemma 5 yields that ψ is equivalent to $x \leq -4$ (using $c' = -5$). □

7.2 Range-Based Quantifier Instantiation

Similarly as in Cooper's elimination algoroithm [23], we can compute the *range* of possible values for a given variable y and instantiating y with all values in the range. For instance, $\exists y(y \leq 2 \wedge 2y \geq 3 \wedge x + 3y = 42)$ can be simplified into $x = 36$.

To obtain the range of possible values of y in the formula φ, we use the formula analysis framework with the following function range_{atom} (whose result is an interval of integers) defined for atomic formulae as follows:

$$\mathsf{range}_{atom}(a \cdot y \leq c, y) = \left(-\infty, \left\lfloor \tfrac{c}{a} \right\rfloor\right] \qquad \text{if } a > 0 \tag{13}$$

$$\mathsf{range}_{atom}(a \cdot y \leq c, y) = \left[\left\lceil \tfrac{c}{a} \right\rceil, +\infty\right) \qquad \text{if } a < 0 \tag{14}$$

$$\mathsf{range}_{atom}(\vec{a} \cdot \vec{x} \leq c, x_i) = \left(-\infty, +\infty\right) \qquad \text{if } \exists j, k \colon j \neq k \wedge a_j \neq 0 \wedge a_k \neq 0 \tag{15}$$

We then employ our formula analysis framework to get the range of y in φ using the function $\mathsf{range}(\varphi, y) = \mathsf{FA}(\varphi, y, \mathsf{range}_{atom}, \cap)$.

Lemma 6. *Let $\psi = \exists y(\varphi(\vec{x}, y))$ be a formula such that $\mathsf{range}(\varphi, y) = [a, b]$ with $a, b \in \mathbb{Z}$. Then ψ is equivalent to the formula $\bigvee_{a \le c \le b} \varphi[y/c]$.*

Proof. It suffices to notice that for all $c \notin [a, b]$ we have $[\![\varphi(\vec{x}, c)]\!] = \emptyset$. □

In our decision procedure, given a formula $\psi = \exists y(\varphi(\vec{x}, y))$, if $\mathsf{range}(\varphi, y) = [a, b]$ for $a, b \in \mathbb{Z}$ and $b - a \le N$ for a parameter N (set by the user), we simplify ψ to $\bigvee_{a \le c \le b} \varphi[y/c]$. In our experiments, we set $N = 0$.

7.3 Modulo Linearization

The next optimization is more complex and helps mainly in practical cases in the benchmarks containing congruences with large moduli. It does not substitute the value of a variable by a constant, but, instead, substitutes a congruence with an equation.

Let $\varphi = \exists y \exists m(\psi \land y + m \equiv_{37} 12)$ such that ψ is 17-best from below w.r.t. y and $\mathsf{range}(\psi, m) = [1, 50]$. Since the modulo constraint $y + m \equiv_{37} 12$ contains two variables (y and m), we cannot use the optimization from Section 7.1. From the modulo constraint and the fact that ψ is 17-best from below w.r.t. y, we can infer that it is sufficient to consider y only in the interval $[-19, 17]$ (we obtained -19 as $17 - 37 + 1$). The reason is that any other y can be mapped to a y' from the same congruence class (modulo 37) that is in the interval and, therefore, gives the same result in the modulo constraint. This, together with the other fact (i.e., $\mathsf{range}(\psi, m) = [1, 50]$) tells us that it is sufficient to only consider the (possibly multiple) linear relations between y and m in the rectangle $[-19, 17] \times [1, 50]$ (cf. Fig. 9). The modulo constraint can, therefore, be substituted by the linear relations to obtain the formula

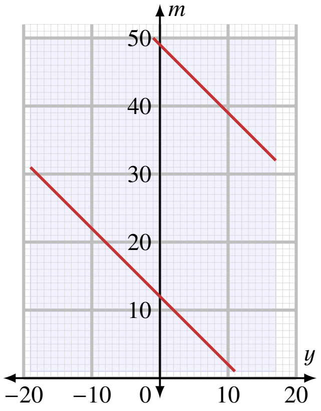


Fig. 9: Modulo linearization.

$$\exists y \exists m(\psi \land ((y \ge -19 \land y \le 11 \land y + m = 12) \lor (y \ge -1 \land y \le 17 \land y + m = 49))). \quad (16)$$

Although the formula seems more complex than the original one, it avoids the large FA to be generated for the modulo constraint (a modulo constraint with $\equiv_k$ needs an FA with k states) and, instead, generates the usually much smaller FAs for the (in)equalities.

The general rewriting rule can be given by the following lemma:

Lemma 7. *Let $\psi(\vec{x}, y, m)$ be a formula s.t. $\mathsf{range}(\psi, m) = [r, s]$ for $r, s \in \mathbb{Z}$, let $\varphi = \exists y \exists m(\psi \land a_y \cdot y + a_m \cdot m \equiv_M k)$, with $a_y \ne 0 \ne a_m$, and $\alpha = \frac{M}{\gcd(a_y, M)}$. If ψ is c-best from below w.r.t. y, then φ is equivalent to the formula*

$$\exists y \exists m \left(\psi \land \left(\bigvee_{i=0}^{N-1} a_y \cdot y + a_m \cdot m = k + (\ell_1 + i) \cdot \alpha\right)\right) \quad (17)$$

where

$$\ell_1 = \begin{cases} \left\lceil \frac{a_y \cdot (c-\alpha+1) + a_m \cdot r - k}{\alpha} \right\rceil & \mathit{for}\ \frac{a_m}{a_y} > 0 \\ \left\lceil \frac{a_y \cdot (c-\alpha+1) + a_m \cdot s - k}{\alpha} \right\rceil & \mathit{for}\ \frac{a_m}{a_y} < 0 \end{cases}, y_1 = \begin{cases} \frac{k + \ell_1 \cdot \alpha - a_m \cdot r}{a_y} & \mathit{for}\ \frac{a_m}{a_y} > 0 \\ \frac{k + \ell_1 \cdot \alpha - a_m \cdot s}{a_y} & \mathit{for}\ \frac{a_m}{a_y} < 0 \end{cases}, \tag{18}$$

and

$$N = \left\lceil \frac{a_y(c - y_1) + a_m(s - r) + 1}{\alpha} \right\rceil. \tag{19}$$

Due to space constraints, we omit a similar lemma for the case when ψ is c-best from above. In our implementation, we use the linearization if the N from Lemma 7 is 1, which is sufficient with many practical cases with large moduli.

8 Experimental Evaluation

We implemented the proposed procedure in a prototype tool called Tool. Tool is written in Python and contains a basic automata library with alphabets encoded using *multi-terminal binary decision diagrams* (MTBDDs), for which it uses the C-based Sylvan library [27] (cf. Appendix B for implementation details). We ran all our experiments on Debian GNU/Linux 12 system with Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz and 32 GiB of RAM with the timeout of 60 s.

Tools. We selected the following tools for comparison: Z3 [60] (version 4.12.2), cvc5 [4] (version 1.0.5), Princess [67] (version 2023-06-19), and Lash [78] (version 0.92). Out of these, only Lash is an automata-based LIA solver; the other tools are general purpose SMT solvers with the LIA theory.

Benchmarks. Our main benchmark comes from SMT-COMP'22 [69], in particular, the categories LIA [70] and NIA (nonlinear integer arithmetic) [71]. We concentrate on formulae in directories `UltimateAutomizer` and (`20190429-`)`UltimateAutomizer Svcomp2019` of these categories (the main difference between LIA and NIA is that LIA formulae are not allowed to use the modulo operator) and remove formulae from NIA that contain multiplication between variables, giving us 372 formulae. We denote this benchmark as `SMT-COMP`. The formulae come from verification of real-world C programs using Ultimate Automizer [44]. Other benchmarks in the categories were omitted. Namely, the `tptp` benchmark is easy for all tools (every tool finished within 1.3 s on each formula). The `psyco` benchmark resembles Boolean reasoning more than integer reasoning. In particular, its formulae contain simple integer constraints (e.g., $x = y + 1$ or just $x = y$) and complex Boolean structure with `ite` operators and quantified Boolean variables. Our prototype tool is not optimized for these features, but with a naive implementation of unwinding of `ite` and with encoding of Boolean variables in a special automaton track, Tool could solve 46 out of the 196 formulae in `psyco`.

Our second benchmark consists of the *Frobenius coin problem* [41] asking the following question: Given a pair of coins of certain coprime denominations a and b, what is the largest number not obtainable as a sum of values of these coins? Or, as a formula,

$$\varphi(p) \overset{\text{def}}{\Leftrightarrow} (\forall x, y \colon p \neq ax + by) \wedge (\forall r (\neg \exists u, v \colon r = au + bv) \Rightarrow r \leq p). \tag{20}$$

Table 1: Comparison of solvers on formulae from the `SMT-COMP` and the `Frobenius` benchmark. Times are given in seconds. The columns **wins** and **losses** show numbers of formulae where Tool performed better and worse (wins/losses caused by timeouts are in parentheses).

	`SMT-COMP` (372)						`Frobenius` (55)					
solver	**timeouts**	**mean**	**median**	**std. dev.**	**wins**	**losses**	**timeouts**	**mean**	**median**	**std. dev.**	**wins**	**losses**
Tool	**17**	1.12	0.26	3.58			**5**	11.79	3.54	16.03		
Tool$_{\text{noopt}}$	73	2.32	0.27	8.16	232 (56)	113 (0)	**5**	11.54	4.06	14.65	27 (0)	21 (0)
Lash	114	3.04	**0.01**	9.94	178 (98)	178 (1)	9	15.72	5.74	20.32	37 (5)	14 (0)
Z3	31	**0.11**	**0.01**	1.35	31 (28)	338 (14)	51	1.66	0.49	2.69	48 (46)	2 (0)
cvc5	28	0.20	0.02	2.42	32 (28)	340 (17)	54	**0.05**	**0.05**	—	49 (49)	1 (0)
Princess	50	4.14	1.14	9.31	354 (40)	8 (7)	13	46.32	45.92	29.03	50 (8)	0 (0)

Each formula is specified by a pair of denominations (a, b), e.g, $(3, 7)$ for which the model is 11. Apart from theoretical interest, the Frobenius coin problem can be used, e.g., for liveness checking of markings of conservative weighted circuits (a variant of Petri nets) [22] or reasoning about automata with counters [76,47,49]. We created a family of 55 formulae encoding the problem with various increasing coin denominations. We denote this benchmark as `Frobenius`. The input format of the benchmarks is SMT-LIB [5], which all tools can handle except Lash—for this, we implemented a simple translator in Tool for translating LIA problems in SMT-LIB into Lash's input format (the time of translation is not included in the runtime of Lash).

Results. We show the results in Table 1. For each benchmark we show the run time statistics together with the number of timeouts and the number of wins/loses for each competitor of Tool (e.g., the value "354 (40)" in the row for Princess in `SMT-COMP` means that Tool was faster than Princess on 354 `SMT-COMP` formulae and in 40 cases out of these, this was because Princess timeouted). Note that statistics about times tend to be biased in favour of tools that timeouted more since the timeouts are not counted.

The first part of the table contains automata-based solvers and the second part contains general SMT solvers. We also measure the effect of our optimizations against Tool$_{\text{noopt}}$, a version of the tool that only performs the classical automata-based procedure from Section 3 without our optimizations.

Discussion. In the comparison with other SMT solvers, from Table 1, automata-based approaches are clearly superior to current SMT solvers on `Frobenius` (confirming the conjecture made in [41]). cvc5 fails already for denominations $(3, 5)$ (where the result is 7) and Z3 follows suite soon; Princess can solve significantly more formulae than Z3 and cvc5, but is still clearly dominated by Tool. See Appendix D for details.

On the `SMT-COMP` benchmark, Tool can solve the most formulae among all tools. It has 17 timeouts, followed by cvc5 with 28 timeouts (out of 372 formulae). On individual examples, the comparison of Tool against Z3 and cvc5 almost always falls under one of the two cases: (i) the solver is one or two orders of magnitude faster than Tool or (ii) the solver times out. This probably corresponds to specific heuristics of Z3 and cvc5 taking effect or not, while Tool has a more robust performance, but is still a prototype and nowhere near as optimized. The performance of Princess is, however, usually much worse. Tool is often complementary to the SMT solvers and was able to solve 6 formulae that no SMT solver did.

Comparison with Tool$_{\text{noopt}}$ shows that the optimizations introduced in this paper have a profound effect on the number of solved cases (which is a proper superset of the cases solved without them). This is most visible on the `SMT-COMP` benchmark, where Tool has 56 TOs less than Tool$_{\text{noopt}}$. On the `Frobenius` benchmark, the results of Tool$_{\text{noopt}}$ and Tool are comparable. Our optimizations had limited impact here since the formulae are built only from a small number of simple atoms (cf. Eq. (20)). In some cases, Tool takes even longer than Tool$_{\text{noopt}}$; this is because the lazy construction explores parts of the state space that would be pruned by the classical construction (e.g., when doing an intersection with a minimized FA with an empty language). This could be possibly solved by algebraic rules tailored for lightweight unsatisfiability checking.

We also tried to evaluate the effect of individual optimizations by selectively turning them off. It turns out that the most critical optimizations are the *simple rewriting rules* (Section 5; when turned off, Tool gave additional 33 timeouts) and *quantifier instantiation* (Section 7; when turned off, Tool gave additional 28 timeouts). On the other hand, surprisingly, turning off *disjunction pruning* (Section 6) did not have a significant effect on the result. By itself (without other optimizations), it can help the basic procedure solve some hard formulae, but its effect is diluted when used with the rest of the optimizations. Still, even though it comes with an additional cost, it still has a sufficient effect to compensate for this overhead.

Comparing with the older automata-based solver Lash, Tool solves more examples in both benchmarks; Lash has 123 TOs in total compared to 22 TOs of Tool. The lower median of Lash on `SMT-COMP` is partially caused by the facts that (i) Lash is a compiled C code while Tool uses a Python frontend, which has a non-negligible overhead and (ii) Lash times out on harder formulae.

Imipact of the individual optimizations

9 Related Work

The decidability of Presburger arithmetic was established already at the beginning of the 20th century by Presburger [61] via *quantifier elimination*. Over time, more efficient quantifier-elimination-based decision procedures occurred, such as the one of Cooper [23] or the one used within the Omega test [62] (which can be seen as a variation of Fourier-Motzkin variable elimination for linear real arithmetic [55, Section 5.4]). The complexity bounds of 2-**NEXP**-hardness and 2-**EXPSPACE** membership for satisfiability checking were obtained by Fischer and Rabin [35] and Berman [6] respectively. Quantifier elimination is often considered impractical due to the blow up in the size of the resulting formula. *Counterexample-guided quantifier instantiation* [63] is a proof-theoretical approach to establish (one-shot) satisfiability of LIA formulae, which can be seen as a lazy version of Cooper's algorithm [23]. It is based on approximating a quantified formula by a set of formulae with the approximation being refined in case it is found too coarse. The approach focuses on formulae with one alternation, but is also extended to any number of alternations (according to the authors, the procedure was implemented in CVC4).

The first *automata-based* decision procedure for Presburger arithmetic can be obtained from Büchi's decision procedure for the second-order logic WS1S [17] by noticing that addition is WS1S-definable. A similar construction for LIA is used by Wolper and Boigelot in [77], except that they avoid performing explicit automata product constructions by using the notion of *concurrent number automata*, which are essentially tuples of synchronized FAs.

Boudet and Comon [13] propose a more direct construction of automata for atomic constraints of the form $a_1x_1 + \ldots + a_nx_n \sim c$ (for $\sim \in \{=, \leq\}$) over natural numbers; we use a construction similar to theirs extended to integers (as used, e.g., in [29]). Moreover, they give a direct construction for a conjunction of equations, which can be seen as a special case of our construction from Section 4. Wolper and Boigelot in [78] discuss optimizations of the procedure from [13] (they use the *most-significant bit first* encoding though), in particular how to remove some states in the construction for automata for inequations based on subsumption obtained syntactically from the formula representing the state (a restricted version of disjunction pruning, cf. Section 6). The works [11,9,12,10] extend the techniques from [78] to solve the mixed *linear integer real arithmetic* (LIRA) using weak Büchi automata, implemented in Lash [1].

WS1S [17] is a closely related logic with an automata-based procedure similar to the one discussed in this paper (as mentioned above, Presburger arithmetic can be encoded into WS1S). The automata-based decision procedure for WS1S is, however, of nonelementary complexity (which is also a lower bound for the logic), it was, however, postulated that the sizes of the obtained automata (when reduced or minimized) describing Presburger-definable sets of integers are bounded by a tower of exponentials of a fixed height. (3-**EXPSPACE**). This postulate was proven by Klaedtke [52] (refined later by Durand-Gasselin and Habermehl [29] who show that all automata during the construction do not exceed size 3-**EXP**). The automata-based decision procedure for WS1S itself has been a subject of extensive study, making many pioneering contributions in the area of automata engineering [39,45,31,53,54,73,33,32,43,34,42], showcasing in the well-known tool Mona [45,31,53,54].

References

1. The Liège automata-based symbolic handler (Lash), `https://people.montefiore.uliege.be/boigelot/research/lash/`
2. Abdulla, P.A., Atig, M.F., Chen, Y., Diep, B.P., Holík, L., Rezine, A., Rümmer, P.: Trau: SMT solver for string constraints. In: Bjørner, N.S., Gurfinkel, A. (eds.) 2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018. pp. 1–5. IEEE (2018). `https://doi.org/10.23919/FMCAD.2018.8602997`, `https://doi.org/10.23919/FMCAD.2018.8602997`
3. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science **155**(2), 291 – 319 (1996). `https://doi.org/https://doi.org/10.1016/0304-3975(95)00182-4`, `http://www.sciencedirect.com/science/article/pii/0304397595001824`
4. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fis-

man, D., Rosu, G. (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. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). `https://doi.org/10.1007/978-3-030-99524-9_24`, `https://doi.org/10.1007/978-3-030-99524-9_24`

5. Barrett, C., Fontaine, P., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2016)
6. Berman, L.: The complexity of logical theories. Theoretical Computer Science **11**(1), 71–77 (1980). `https://doi.org/10.1016/0304-3975(80)90037-7`
7. Berzish, M., Day, J.D., Ganesh, V., Kulczynski, M., Manea, F., Mora, F., Nowotka, D.: Towards more efficient methods for solving regular-expression heavy string constraints. Theor. Comput. Sci. **943**, 50–72 (2023). `https://doi.org/10.1016/j.tcs.2022.12.009`, `https://doi.org/10.1016/j.tcs.2022.12.009`
8. Blahoudek, F., Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Word equations in synergy with regular constraints. In: Proc. of FM'23. Springer (2023)
9. Boigelot, B., Jodogne, S., Wolper, P.: On the use of weak automata for deciding linear arithmetic with integer and real variables. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2083, pp. 611–625. Springer (2001). `https://doi.org/10.1007/3-540-45744-5_50`, `https://doi.org/10.1007/3-540-45744-5_50`
10. Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Trans. Comput. Log. **6**(3), 614–633 (2005). `https://doi.org/10.1145/1071596.1071601`, `https://doi.org/10.1145/1071596.1071601`
11. Boigelot, B., Rassart, S., Wolper, P.: On the expressiveness of real and integer arithmetic automata (extended abstract). In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1443, pp. 152–163. Springer (1998). `https://doi.org/10.1007/BFb0055049`, `https://doi.org/10.1007/BFb0055049`
12. Boigelot, B., Wolper, P.: Representing arithmetic constraints with finite automata: An overview. In: Stuckey, P.J. (ed.) Logic Programming, 18th International Conference, ICLP 2002, Copenhagen, Denmark, July 29 - August 1, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2401, pp. 1–19. Springer (2002). `https://doi.org/10.1007/3-540-45619-8_1`, `https://doi.org/10.1007/3-540-45619-8_1`
13. Boudet, A., Comon, H.: Diophantine equations, Presburger arithmetic and finite automata. In: Kirchner, H. (ed.) Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1059, pp. 30–43. Springer (1996). `https://doi.org/10.1007/3-540-61064-2_27`, `https://doi.org/10.1007/3-540-61064-2_27`
14. Brzozowski, J.A.: Canonical regular expressions and minimal state graphs for definite events. In: Proc. Sympos. Math. Theory of Automata (New York, 1962), Microwave Research Institute Symposia Series, vol. Vol. XII, pp. 529–561. Polytechnic, Brooklyn, NY (1963)
15. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964). `https://doi.org/10.1145/321239.321249`, `http://doi.acm.org/10.1145/321239.321249`
16. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. of International Congress on Logic, Method, and Philosophy of Science 1960. Stanford Univ. Press, Stanford (1962)

17. Büchi, J.R.: Weak second-order arithmetic and finite automata. Zeitscrift fur mathematische Logic und Grundlagen der Mathematik **6**, 66–92 (1960)
18. Chen, T., Flores-Lamas, A., Hague, M., Han, Z., Hu, D., Kan, S., Lin, A.W., Rümmer, P., Wu, Z.: Solving string constraints with regex-dependent functions through transducers with priorities and variables. Proc. ACM Program. Lang. **6**(POPL), 1–31 (2022). `https://doi.org/10.1145/3498707`, `https://doi.org/10.1145/3498707`
19. Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Solving string constraints with lengths by stabilization. Proc. ACM Program. Lang. **7**(OOPSLA2) (oct 2023). `https://doi.org/10.1145/3622872`
20. Chen, Y.F., Chocholatý, D., Havlena, V., Holík, L., Lengál, O., Síč, J.: Z3-noodler: An automata-based string solver (technical report). CoRR **abs/2310.08327** (2023). `https://doi.org/10.48550/arXiv.2310.08327`, `https://doi.org/10.48550/arXiv.2310.08327`
21. Chocholatý, D., Fiedor, T., Havlena, V., Holík, L., Hruška, M., Lengál, O., Síč, J.: Mata: A fast and simple finite automata library (technical report). CoRR **abs/2310.10136** (2023). `https://doi.org/10.48550/arXiv.2310.10136`, `https://doi.org/10.48550/arXiv.2310.10136`, To appear at TACAS'23
22. Chrzastowski-Wachtel, P., Raczunas, M.: Liveness of weighted circuits and the diophantine problem of frobenius. In: Ésik, Z. (ed.) Fundamentals of Computation Theory. pp. 171–180. Springer Berlin Heidelberg, Berlin, Heidelberg (1993)
23. Cooper, D.: Theorem proving in arithmetic without multiplication. Machine Intelligence **7**, 91–99 (1972)
24. Cox, A., Leasure, J.: Model checking regular language constraints. CoRR **abs/1708.09073** (2017)
25. Dantzig, G.B.: Inductive proof of the simplex method. IBM J. Res. Dev. **4**(5), 505–506 (1960). `https://doi.org/10.1147/RD.45.0505`, `https://doi.org/10.1147/rd.45.0505`
26. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: A theorem prover for program checking. J. ACM **52**(3), 365–473 (may 2005). `https://doi.org/10.1145/1066100.1066102`, `https://doi.org/10.1145/1066100.1066102`
27. van Dijk, T., van de Pol, J.: Sylvan: multi-core framework for decision diagrams. Int. J. Softw. Tools Technol. Transf. **19**(6), 675–696 (2017). `https://doi.org/10.1007/s10009-016-0433-2`, `https://doi.org/10.1007/s10009-016-0433-2`
28. Doyen, L., Raskin, J.: Antichain algorithms for finite automata. In: Esparza, J., Majumdar, R. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6015, pp. 2–22. Springer (2010). `https://doi.org/10.1007/978-3-642-12002-2_2`, `https://doi.org/10.1007/978-3-642-12002-2_2`
29. Durand-Gasselin, A., Habermehl, P.: On the use of non-deterministic automata for Presburger arithmetic. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6269, pp. 373–387. Springer (2010). `https://doi.org/10.1007/978-3-642-15375-4_26`, `https://doi.org/10.1007/978-3-642-15375-4_26`
30. Egly, U.: On the value of antiprenexing. In: Pfenning, F. (ed.) Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Kiev, Ukraine, July 16-22, 1994, Proceedings. Lecture Notes in Computer Science, vol. 822, pp. 69–83. Springer (1994). `https://doi.org/10.1007/3-540-58216-9_30`, `https://doi.org/10.1007/3-540-58216-9_30`

31. Elgaard, J., Klarlund, N., Møller, A.: Mona 1.x: new techniques for WS1S and WS2S. In: CAV 1998. Lecture Notes in Computer Science, vol. 1427, pp. 516–520. BRICS, Department of Computer Science, Aarhus University, Springer (1998)
32. Fiedor, T., Holík, L., Janku, P., Lengál, O., Vojnar, T.: Lazy automata techniques for WS1S. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10205, pp. 407–425 (2017). https://doi.org/10.1007/978-3-662-54577-5_24, https://doi.org/10.1007/978-3-662-54577-5_24
33. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. In: Baier, C., Tinelli, C. (eds.) 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. Lecture Notes in Computer Science, vol. 9035, pp. 658–674. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_59, https://doi.org/10.1007/978-3-662-46681-0_59
34. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. Acta Informatica **56**(3), 205–228 (2019). https://doi.org/10.1007/s00236-018-0331-z, https://doi.org/10.1007/s00236-018-0331-z
35. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of Presburger arithmetic. In: Proceedings of the SIAM-AMS Symposium in Applied Mathematics. vol. 7, pp. 27—41 (1974)
36. Ge, Y., de Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: Bouajjani, A., Maler, O. (eds.) Computer Aided Verification. pp. 306–320. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
37. Ghilardi, S., Nicolini, E., Ranise, S., Zucchelli, D.: Decision procedures for extensions of the theory of arrays. Ann. Math. Artif. Intell. **50**(3-4), 231–254 (2007). https://doi.org/10.1007/s10472-007-9078-x, https://doi.org/10.1007/s10472-007-9078-x
38. van Glabbeek, R.J., Ploeger, B.: Five determinisation algorithms. In: Ibarra, O.H., Ravikumar, B. (eds.) Implementation and Applications of Automata, 13th International Conference, CIAA 2008, San Francisco, California, USA, July 21-24, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5148, pp. 161–170. Springer (2008). https://doi.org/10.1007/978-3-540-70844-5_17, https://doi.org/10.1007/978-3-540-70844-5_17
39. Glenn, J., Gasarch, W.: Implementing WS1S via finite automata. In: Raymond, D.R., Wood, D., Yu, S. (eds.) Workshop on Implementing Automata. Lecture Notes in Computer Science, vol. 1260, pp. 50–63. Springer (1996)
40. Google: RE2, https://github.com/google/re2
41. Haase, C.: A survival guide to Presburger arithmetic. ACM SIGLOG News **5**(3), 67–82 (2018). https://doi.org/10.1145/3242953.3242964, https://doi.org/10.1145/3242953.3242964
42. Havlena, V., Holík, L., Lengál, O., Vales, O., Vojnar, T.: Antiprenexing for WSkS: A little goes a long way. In: Albert, E., Kovács, L. (eds.) LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020. EPiC Series in Computing, vol. 73, pp. 298–316. EasyChair (2020). https://doi.org/10.29007/6bfc, https://doi.org/10.29007/6bfc
43. Havlena, V., Holík, L., Lengál, O., Vojnar, T.: Automata terms in a lazy WSkS decision procedure. In: Fontaine, P. (ed.) Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11716, pp. 300–318. Springer (2019).

https://doi.org/10.1007/978-3-030-29436-6_18, https://doi.org/10.1007/978-3-030-29436-6_18

44. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 36–52. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_2, https://doi.org/10.1007/978-3-642-39799-8_2
45. Henriksen, J.G., Jensen, J.L., Jørgensen, M.E., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: Mona: Monadic second-order logic in practice. In: Brinksma, E., Cleaveland, R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) TACAS'95. Lecture Notes in Computer Science, vol. 1019, pp. 89–110. Springer (1995)
46. Hieronymi, P., Ma, D., Oei, R., Schaeffer, L., Schulz, C., Shallit, J.O.: Decidability for Sturmian words. In: Manea, F., Simpson, A. (eds.) 30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference). LIPIcs, vol. 216, pp. 24:1–24:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPICS.CSL.2022.24, https://doi.org/10.4230/LIPIcs.CSL.2022.24
47. Holík, L., Lengál, O., Saarikivi, O., Turonová, L., Veanes, M., Vojnar, T.: Succinct determinisation of counting automata via sphere construction. In: Lin, A.W. (ed.) Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11893, pp. 468–489. Springer (2019). https://doi.org/10.1007/978-3-030-34175-6_24, https://doi.org/10.1007/978-3-030-34175-6_24
48. Hopcroft, J.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971), pp. 189–196. Academic Press, New York-London (1971)
49. Hu, D., Wu, Z.: String constraints with regex-counting and string-length solved more efficiently. In: Hermanns, H., Sun, J., Bu, L. (eds.) Dependable Software Engineering. Theories, Tools, and Applications - 9th International Symposium, SETTA 2023, Nanjing, China, November 27-29, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14464, pp. 1–20. Springer (2023). https://doi.org/10.1007/978-981-99-8664-4_1, https://doi.org/10.1007/978-981-99-8664-4_1
50. Hyperscan.io: https://www.hyperscan.io/ (2021)
51. Jonás, M., Strejcek, J.: Q3B: an efficient bdd-based SMT solver for quantified bit-vectors. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 64–73. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_4, https://doi.org/10.1007/978-3-030-25543-5_4
52. Klaedtke, F.: Bounds on the automata size for presburger arithmetic. ACM Trans. Comput. Log. **9**(2), 11:1–11:34 (2008). https://doi.org/10.1145/1342991.1342995, https://doi.org/10.1145/1342991.1342995
53. Klarlund, N.: A theory of restrictions for logics and automata. In: Proc. of CAV'99. LNCS, vol. 1633, pp. 406–417. Springer (1999)
54. Klarlund, N., Møller, A., Schwartzbach, M.I.: Mona implementation secrets. International Journal of Foundations of Computer Science **13**(4), 571–586 (2002)
55. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer (2016). https://doi.org/10.1007/978-3-662-50497-0, https://doi.org/10.1007/978-3-662-50497-0

56. Kuncak, V., Mayer, M., Piskac, R., Suter, P.: Software synthesis procedures. Commun. ACM **55**(2), 103–111 (2012). https://doi.org/10.1145/2076450.2076472, https://doi.org/10.1145/2076450.2076472
57. Kuper, G.M., Libkin, L., Paredaens, J. (eds.): Constraint Databases. Springer (2000). https://doi.org/10.1007/978-3-662-04031-7, https://doi.org/10.1007/978-3-662-04031-7
58. Monniaux, D.: Automatic modular abstractions for linear constraints. In: Shao, Z., Pierce, B.C. (eds.) Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009. pp. 140–151. ACM (2009). https://doi.org/10.1145/1480881.1480899, https://doi.org/10.1145/1480881.1480899
59. Moseley, D., Nishio, M., Rodriguez, J.P., Saarikivi, O., Toub, S., Veanes, M., Wan, T., Xu, E.: Derivative based nonbacktracking real-world regex matching with backtracking semantics. Proc. ACM Program. Lang. **7**(PLDI), 1026–1049 (2023). https://doi.org/10.1145/3591262, https://doi.org/10.1145/3591262
60. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24
61. Presburger, M.: Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. In: Comptes Rendus du I congrès de Mathématiciens des Pays Slaves. pp. 92—101 (1929)
62. Pugh, W.W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. In: Martin, J.L. (ed.) Proceedings Supercomputing '91, Albuquerque, NM, USA, November 18-22, 1991. pp. 4–13. ACM (1991). https://doi.org/10.1145/125826.125848, https://doi.org/10.1145/125826.125848
63. Reynolds, A., King, T., Kuncak, V.: Solving quantified linear arithmetic by counterexample-guided instantiation. Formal Methods Syst. Des. **51**(3), 500–532 (2017). https://doi.org/10.1007/s10703-017-0290-y, https://doi.org/10.1007/s10703-017-0290-y
64. Reynolds, A., Kuncak, V., Tinelli, C., Barrett, C.W., Deters, M.: Refutation-based synthesis in SMT. Formal Methods Syst. Des. **55**(2), 73–102 (2019). https://doi.org/10.1007/S10703-017-0270-2, https://doi.org/10.1007/s10703-017-0270-2
65. Reynolds, A., Tinelli, C., de Moura, L.: Finding conflicting instances of quantified formulas in smt. In: Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design. p. 195–202. FMCAD '14, FMCAD Inc, Austin, Texas (2014)
66. Robinson, J.A., Voronkov, A. (eds.): Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001), https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning
67. Rümmer, P.: A constraint sequent calculus for first-order logic with linear integer arithmetic. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5330, pp. 274–289. Springer (2008). https://doi.org/10.1007/978-3-540-89439-1_20, https://doi.org/10.1007/978-3-540-89439-1_20
68. Schüle, T., Schneider, K.: Verification of data paths using unbounded integers: Automata strike back. In: Bin, E., Ziv, A., Ur, S. (eds.) Hardware and Software, Verification and Testing,

Second International Haifa Verification Conference, HVC 2006, Haifa, Israel, October 23-26, 2006. Revised Selected Papers. Lecture Notes in Computer Science, vol. 4383, pp. 65–80. Springer (2006). https://doi.org/10.1007/978-3-540-70889-6_5, https://doi.org/10.1007/978-3-540-70889-6_5

69. SMT-COMP'22: https://smt-comp.github.io/2022/ (2022)
70. SMT-LIB: LIA. https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/LIA (2023)
71. SMT-LIB: NIA. https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/NIA (2023)
72. Stanford, C., Veanes, M., Bjørner, N.: Symbolic boolean derivatives for efficiently solving extended regular expression constraints. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. p. 620–635. PLDI 2021, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3453483.3454066, https://doi.org/10.1145/3453483.3454066
73. Topnik, C., Wilhelm, E., Margaria, T., Steffen, B.: jMosel: A stand-alone tool and jABC plugin for M2L(Str). In: Valmari, A. (ed.) 13th International SPIN Workshop. Lecture Notes in Computer Science, vol. 3925, pp. 293–298. Springer Berlin Heidelberg (2006). https://doi.org/10.1007/11691617_18, http://dx.doi.org/10.1007/11691617_18
74. Traytel, D.: A coalgebraic decision procedure for WS1S. In: Kreutzer, S. (ed.) 24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany. LIPIcs, vol. 41, pp. 487–503. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015). https://doi.org/10.4230/LIPIcs.CSL.2015.487, https://doi.org/10.4230/LIPIcs.CSL.2015.487
75. Traytel, D., Nipkow, T.: Verified decision procedures for MSO on words based on derivatives of regular expressions. J. Funct. Program. **25** (2015). https://doi.org/10.1017/S0956796815000246, https://doi.org/10.1017/S0956796815000246
76. Turonová, L., Holík, L., Lengál, O., Saarikivi, O., Veanes, M., Vojnar, T.: Regex matching with counting-set automata. Proc. ACM Program. Lang. **4**(OOPSLA), 218:1–218:30 (2020). https://doi.org/10.1145/3428286, https://doi.org/10.1145/3428286
77. Wolper, P., Boigelot, B.: An automata-theoretic approach to Presburger arithmetic constraints (extended abstract). In: Mycroft, A. (ed.) Static Analysis, Second International Symposium, SAS'95, Glasgow, UK, September 25-27, 1995, Proceedings. Lecture Notes in Computer Science, vol. 983, pp. 21–32. Springer (1995). https://doi.org/10.1007/3-540-60360-3_30, https://doi.org/10.1007/3-540-60360-3_30
78. Wolper, P., Boigelot, B.: On the construction of automata from linear arithmetic constraints. In: Graf, S., Schwartzbach, M.I. (eds.) Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1785, pp. 1–19. Springer (2000). https://doi.org/10.1007/3-540-46419-0_1, https://doi.org/10.1007/3-540-46419-0_1

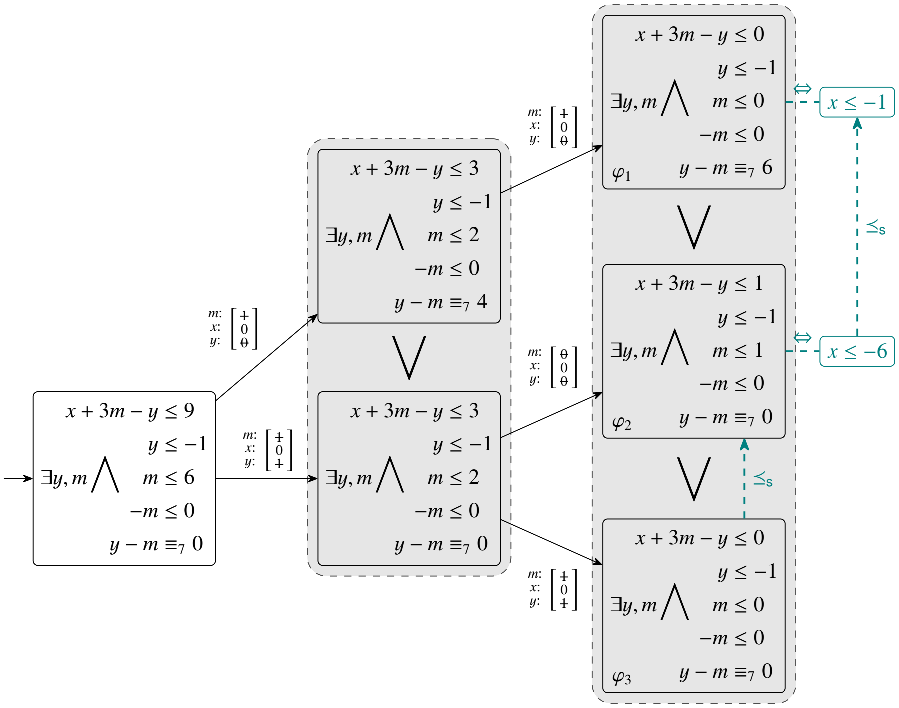


Fig. 10: Fragment of the generated space for the formula in the example.

A More Comprehensive Example of Our Optimizations

Consider the formula

$$\exists y, m(x + 3m - y \le 9 \wedge y \le -1 \wedge m \le 6 \wedge -m \le 0 \wedge y - m \equiv_7 0) \tag{21}$$

and see Fig. 10 for a part of the generated FA (for simplicity, we only consider a fragment of the constructed automaton to demonstrate our technique).

Let us focus on the configuration after reading the word x: [0][0]: $\varphi_1 \vee \varphi_2 \vee \varphi_3$. First, we examine the relation between φ_2 and φ_3. We notice that the two formulae look similar with the only difference being in two pairs of atoms: $x + 3m - y \le 1$ and $x + 3m - y \le 0$, and $m \le 1$ and $m \le 0$ respectively. Since $0 \le 1$ there are structural subsumptions $x + 3m - y \le 0 \preceq_s x + 3m - y \le 1$ and $m \le 0 \preceq_s m \le 1$, which yields $\varphi_3 \preceq_s \varphi_2$, and we can therefore use disjunction pruning (Section 6) to simplify $\varphi_1 \vee \varphi_2 \vee \varphi_3$ to $\varphi_1 \vee \varphi_2$.

Next, we analyze $\varphi_1 = \exists y, m(\psi_1)$. First, we compute $\mathsf{range}(\psi_1, m) = [0, 0]$ and, based on that, perform the substitution $\psi_1[m/0]$, obtaining (after simplifications) the formula $\psi'_1 = x - y \le 0 \wedge y \le -1 \wedge y \equiv_7 6$. Then, we analyze the behaviour of y in ψ'_1 by computing $\mathsf{blw}(x - y \le 0 \wedge y \le -1, y) = -1$. Based on this, we know that $x - y \le 0 \wedge y \le -1$ is (-1)-best from below, so we can use Lemma 5 to instantiate y

in ψ_1' with -5 (the largest number less than -1 satisfying the modulo constraint), obtaining the (quantifier-free) formula $x \leq -1$.

Finally, we focus on $\varphi_2 = \exists y, m(\psi_2)$ again. First, we compute $\mathsf{range}(\psi_2, m) = [0, 1]$ and rewrite φ_2 to $\exists y(\psi_2[m/0] \vee \psi_2[m/1])$. After antiprenexing, this will be changed to $\exists y(\psi_2[m/0]) \vee \exists y(\psi_2[m/1])$. Using similar reasoning as in the previous paragraph, we can analyze the two disjuncts in the formula to obtain the formula $x \leq -6 \vee x \leq -8$. With disjunction pruning, we obtain the final result of simplification of φ_2 as the formula $x \leq -6$. In the end, again using disjunction pruning, the whole formula $\varphi_1 \vee \varphi_2 \vee \varphi_3$ can be simplified to $x \leq -1$.

B Implementation Details

Tool's is architecturally divided into a frontend written in Python responsible for preprocessing the input formula and orchestrating the decision procedure, and two user-selectable backends providing the frontend with automata operations. The first of Tool's backends is implemented in Python and is geared towards experimentation while sacrificing execution speed. The backend is capable of tracking the origin of automata states all the way to the atomic constraints, allowing analysis of phenomena in solver's runtime traces, e.g., situations when applying minimization removing large number of automata states. Having rich information thus allows one to use impactful minimization as a natural guide for finding new formula rewriting rules. The backend represents transitions symbols explicitly, avoiding cumbersome propagation of state-origin information that happens when manipulating automata that represent the transition function symbolically.

Tool's second backend is written in C++ and provides algorithms for automata with their transition function represented symbolically using *multi-terminal binary decision diagrams* (MTBDDs), for which it uses the Sylvan library [27]. The need for a symbolic representation is natural, as the size of the automaton alphabet grows exponentially with the number of variables. As Sylvan is implemented in C, it would be possible to write a small wrapper providing Tool's frontend with low-level MTBDD manipulation (actually, a previous version of Tool was using such a wrapper). However, this approach quickly becomes impractical, as a nontrivial amount of information needs to be exchanged between the frontend and the low-level MTBDD operations. Therefore, Tool's MTBDD backend is written entirely in C++ and exposes only high-level API calls manipulating entire automata.

C Proofs

Lemma 3. *For LIA formulae φ_1 and φ_2, if $\varphi_1 \preceq_{\mathsf{s}} \varphi_2$, then $\varphi_1 \Rightarrow \varphi_2$.*

Proof. By induction.

- Base case: The only interesting base case is $\vec{a} \cdot \vec{x} \leq c_1 \preceq_{\mathsf{s}} \vec{a} \cdot \vec{x} \leq c_2$. For showing that if $c_1 \leq c_2$, then $[\![\vec{a} \cdot \vec{x} \leq c_1]\!] \subseteq [\![\vec{a} \cdot \vec{x} \leq c_2]\!]$, it suffices to notice that every model of the formula on the left-hand side is also a model of the formula on the right-hand side.

- Inductive case: All inductive cases are easy to prove by reasoning over the models of the formulae. □

Lemma 4. *Let $c \in \mathbb{Z} \cup \{\pm\infty\}$ and $\varphi(\vec{x}, y)$ be a formula c-monotone w.r.t. y such that $\varphi[y/c]$ is defined. Then the formula $\exists y(\varphi(\vec{x}, y))$ is equivalent to the formula $\varphi[y/c]$.*

Proof. The case for $c \in \mathbb{Z}$ follows directly from the definition of monotonicity. For the case of $c = \pm\infty$, we prove only the case of $c = +\infty$ and φ being c-best from below wrt. y (the other case can be proven analogously). From the definition of best from below formulae, we have that $\llbracket \varphi(\vec{x}, y_1) \rrbracket \subseteq \llbracket \varphi(\vec{x}, y_2) \rrbracket$ for each $y_1 \leq y_2$. Since $\varphi[y/\infty]$ is defined, let A be atoms of φ s.t. the substitution $y/+\infty$ yields $\top$ or $\bot$. These are all atoms containing with y as a free variable. Further consider the function

$$f(\vec{a} \cdot \vec{x} + a_y \cdot y \leq c) = \frac{c - \vec{a} \cdot \vec{x}}{a_y}$$

and based on it the maximum for $\vec{x}$ as

$$g(\vec{x}) = \max\{f(\psi)(\vec{x}) + 1 \mid \psi \in A\}.$$

Then, we have that $\varphi[y/\infty] = \varphi(\vec{x}, g(\vec{x}))$ and moreover $\varphi[y/\infty] = \varphi(\vec{x}, y)$ for each $y \geq g(\vec{x})$. which implies that $\llbracket \varphi[y/\infty] \rrbracket \subseteq \llbracket \exists y(\varphi(\vec{x}, y)) \rrbracket$. Now consider a model ν of the formula $\exists y(\varphi(\vec{x}, y))$. Then, there is a model $\nu' = \nu \cup \{y \mapsto \ell\}$. Since $\ell < \infty$, from the monotonicity of φ and from the reasoning above we get that ν is a model of $\varphi[y/\infty]$, which concludes the proof. □

Lemma 5. *Let $c \in \mathbb{Z}$ and $\varphi(\vec{x}, y)$ be a formula c-monotone w.r.t. y for $c \in \mathbb{Z}$. Then, the formula $\exists y(\varphi(\vec{x}, y) \wedge y \equiv_m k)$ is equivalent to the formula $\varphi[y/c']$ where (i) $c' = \max\{\ell \in \mathbb{Z} \mid \ell \equiv_m k, \ell \leq c\}$ if φ is c-best from below, and (ii) $c' = \min\{\ell \in \mathbb{Z} \mid \ell \equiv_m k, \ell \geq c\}$ if φ is c-best from above.*

Proof. We prove the c-best-from-below case only. The other case can be proven analogously. First, we show that each model ν of $\varphi(\vec{x}, c')$ is also a model of $\exists y(\varphi(\vec{x}, y) \wedge y \equiv_m k)$. Consider $\nu' = \nu \cup \{y \mapsto c'\}$. Since $c' \equiv_m k$, we have that ν' is a model of $\varphi(\vec{x}, y) \wedge y \equiv_m k$.

Now, we prove that each model ν of $\exists y(\varphi(\vec{x}, y) \wedge y \equiv_m k)$ is also a model of $\varphi(\vec{x}, c')$. Since ν is a model of the first formula, we have that there is some ℓ s.t. $\nu' = \nu \cup \{y \mapsto \ell\}$ is a model of $\varphi(\vec{x}, y) \wedge y \equiv_m k$. Since φ is c-best from below w.r.t. y, we have that $\ell \leq c$. Secondly, from the second conjunct we get that $\ell \equiv_m k$. From the definition of c' we have that $\ell \leq c'$. Finally, if we again use the property of best-from-belowness of φ together with $\ell \leq c'$, we obtain that ν is a model of $\varphi(\vec{x}, c')$. □

Lemma 8. *For $a_y, a_m, m, k, c \in \mathbb{Z}$ and $M \in \mathbb{Z}^+$, if there is $y_1 \in \mathbb{Z}$ such that $a_y \cdot y_1 + a_m \cdot m \equiv_M k$, then there exist $y_2 \in [c - \alpha + 1, c]$ and $y_3 \in [c, c + \alpha - 1]$ such that $a_y \cdot y_2 + a_m \cdot m \equiv_M k$ and $a_y \cdot y_3 + a_m \cdot m \equiv_M k$ for $\alpha = \frac{M}{\gcd(a_y, M)}$.*

Proof. Assume that $a_y \cdot y_1 + a_m \cdot m \equiv_M k$ and let us show how to choose y_2 from the interval such that $a_y \cdot y_2 + a_m \cdot m \equiv_M k$. Rewriting the two modulo constraints into equations, we get the following two linear constraints,

$$a_y \cdot y_1 + a_m \cdot m + d_1 \cdot M = k \tag{22}$$

$$a_y \cdot y_2 + a_m \cdot m + d_2 \cdot M = k, \tag{23}$$

for some $d_1, d_2 \in \mathbb{Z}$. From equality of the left-hand sides, we obtain

$$a_y \cdot y_1 + d_1 \cdot M = a_y \cdot y_2 + d_2 \cdot M \tag{24}$$

and, after rearrangement,

$$a_y \cdot y_1 - a_y \cdot y_2 = d_2 \cdot M - d_1 \cdot M, \tag{25}$$

which is equivalent to

$$a_y(y_1 - y_2) = M(d_2 - d_1). \tag{26}$$

Since d_2 and d_1 are both existentially quantified, we can substitute $d_2 - d_1$ by an existentially quanfitied variable d, obtaining

$$a_y(y_1 - y_2) = M \cdot d. \tag{27}$$

Note that since we are reasoning over integers, we cannot just divide the right-hand side by a_y. Instead, let $\beta = \gcd(a_y, M)$. Then we can express a_y and M as $a_y = a'_y \cdot \beta$ and $M = M' \cdot \beta$, so

$$a'_y \cdot \beta \cdot (y_1 - y_2) = M' \cdot \beta \cdot d. \tag{28}$$

The β's cancel out, so we get

$$a'_y(y_1 - y_2) = M' \cdot d \tag{29}$$

where a'_y and M' are coprime. Because of this, the equation can only hold if d is a multiple of a'_y, i.e., $d = h \cdot a'_y$ for some $h \in \mathbb{Z}$:

$$a'_y(y_1 - y_2) = M' \cdot h \cdot a'_y \tag{30}$$

Now, a'_y's cancel out, i.e.,

$$y_1 - y_2 = M' \cdot h \qquad \Longrightarrow \qquad y_2 = y_1 - M' \cdot h. \tag{31}$$

Because $M = M' \cdot \gcd(a_y, M)$, we have that $M' = \frac{M}{\gcd(a_y,M)}$, which is the same as the definition of α. Therefore, for any interval of the size at least α, there will be a value of h such that $y_2 = y_1 - \alpha \cdot h$ is in the interval. We can duplicate similar reasoning for y_3. □

Lemma 7. *Let $\psi(\vec{x}, y, m)$ be a formula s.t.* $\mathsf{range}(\psi, m) = [r, s]$ *for $r, s \in \mathbb{Z}$, let $\varphi = \exists y \exists m(\psi \land a_y \cdot y + a_m \cdot m \equiv_M k)$, with $a_y \neq 0 \neq a_m$, and $\alpha = \frac{M}{\gcd(a_y,M)}$. If ψ is c-best from below w.r.t. y, then φ is equivalent to the formula*

$$\exists y \exists m \left(\psi \land \left(\bigvee_{i=0}^{N-1} a_y \cdot y + a_m \cdot m = k + (\ell_1 + i) \cdot \alpha\right)\right) \tag{17}$$

where

$$\ell_1 = \begin{cases} \left\lceil \frac{a_y \cdot (c-\alpha+1) + a_m \cdot r - k}{\alpha} \right\rceil & \mathit{for}\ \frac{a_m}{a_y} > 0 \\ \left\lceil \frac{a_y \cdot (c-\alpha+1) + a_m \cdot s - k}{\alpha} \right\rceil & \mathit{for}\ \frac{a_m}{a_y} < 0 \end{cases}, y_1 = \begin{cases} \frac{k + \ell_1 \cdot \alpha - a_m \cdot r}{a_y} & \mathit{for}\ \frac{a_m}{a_y} > 0 \\ \frac{k + \ell_1 \cdot \alpha - a_m \cdot s}{a_y} & \mathit{for}\ \frac{a_m}{a_y} < 0 \end{cases}, \tag{18}$$

and

$$N = \left\lceil \frac{a_y(c - y_1) + a_m(s - r) + 1}{\alpha} \right\rceil. \tag{19}$$

Proof. From Lemma 8 and the fact that ψ is c-decreasing w.r.t. y, it follows that it is sufficient to consider y in the interval $[c - \alpha + 1, c]$. Therefore, for linearizing the modulo constraint $a_y \cdot y + a_m \cdot m \equiv_M k$ into a disjunction of linear equations of the form $a_y \cdot y + a_m \cdot m = k + \ell_j \cdot \alpha$ for different ℓ_j that intersect the rectangle $[c-\alpha+1, c] \times [r, s]$, we need to (a) find the ℓ_1 of the left-most line intersecting the rectangle (we assume y is on the horizontal axis and m is on the vertical axis) and (b) find the number N of lines intersecting the rectangle. The value of ℓ_1 is computed using a technical excursion into high school analytic geometry that we do not reproduce here (we just note that the case split is to distinguish lines representing increasing/decreasing functions). From ℓ_1, we compute the y-coordinate of the left-most line intersecting the rectangle, denoted as y_1, and also N, both by using analytic geometry one more time. □

D Additional Plots

In this section, we present additional plots underlying results presented in Section 8. In particular, in Fig. 11 we show scatter plots comparing Tool with other automata-based tools and in Fig. 12 we show comparison of Tool with other tools on `Frobenius`.

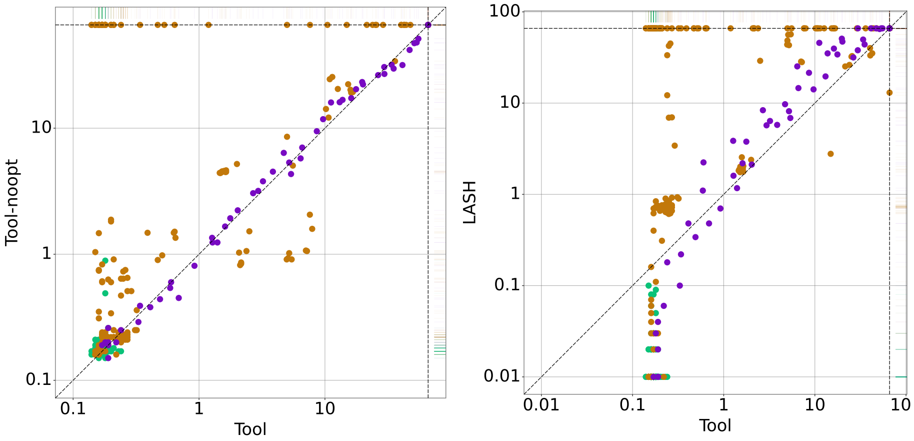


Fig. 11: Comparison of automata-based LIA solvers on formulae from SMT-COMP: ● `UltimateAutomizer` (153), ● `UltimateAutomizerSvcomp2019` (219) and ● `Frobenius Coin Problem` (55). Times are in seconds, axes are logarithmic. Dashed lines represent timeouts (60 s).

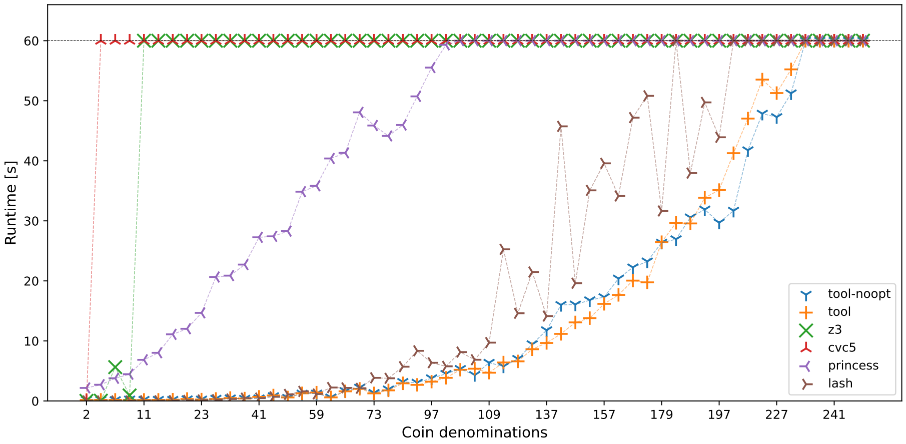


Fig. 12: Run times of solvers on `Frobenius`.