# Presburger Arithmetic

Nhat Minh Hoang

Technische Universität München
hoang@in.tum.de

**Abstract.** In this paper a procedure for constructing an automaton that decides Presburger arithmetic is explained. Additionally, the construction will be extended to accept integer solutions and also be specialized for equations, strict inequalities, divisibility relations and linear systems of equations.

## 1 Introduction

The problem of solving linear constraints is one of the most studied and well-known problems in computer science. Finding useful applications is not a difficult task since a multitude of problems can be expressed as linear programs. Especially optimization problems are very well suited for linear programming such as various network flow problems can be expressed as linear constraints. The specialized version of linear programming, integer programming, is a much harder task to solve since it restricts its solution space to integers only. The methods for solving integer programming integer programs often search for an optimal solution of the constraint and can usually be divided in exact methods and heuristic methods such as hill climbing [21], linear programming relaxation [16], cutting planes [14] or simulated annealing [18].

Restricting the solutions of a linear constraint to natural numbers yields an instance of Presburger arithmetic [19]. Presburger arithmetic is a First-Order theory over the natural numbers. This logic only allows for addition and also multiplication by a constant. Presburger arithmetic is weaker than Peano arithmetic regarding the expressiveness since Presburger arithmetic does not allow for multiplication with non-constant variables. However, Presburger arithmetic is decidable in contrast to Peano arithmetic. Therefore, a number of decision procedures exists that decide for all Presburger formulae

Mojżesz Presburger showed that his logic is decidable [19] by proving that it admits quantifier elimination. Later on, J.R. Büchi proved the decidability of the Second-Order Monadic Logic with one successor [4, 3]. It includes that every logic definable in Weak Second-Order Monadic Logic with one successor WS1S is decidable using finite automata. It was also proven that Presburger arithmetic is definable in WS1S and therefore it can be decided using finite automata.

Boudet and Comon [1] use Büchi's technique for constructing the automaton and use this construction to first solve linear Diophantine equations. Then it was extended to solve Presburger arithmetic through intersection, union and negation of the regular languages. In this paper the construction by Boudet and Comon [1] will be presented as well as introducing other meaningful extensions to the procedure.

## 2 Presburger Arithmetic

Presburger arithmetic is a First-Order theory reasoning about addition where 0 and 1 are the only constants.

### 2.1 Syntax

The following formal grammars lay out the basic definition of Presburger arithmetic.

**Definition 1 (Terms).** *Let VAR be a finite set of variables. Then the following formal grammar defines Presburger-Arithmetic* ***terms****:*

$$t ::= \mathbb{N}_0 \mid \mathit{VAR} \mid t + t \mid \mathbb{N}_0 * t$$

**Definition 2 (Formula).** *An* ***atomic formula*** *consists of two terms without negation, logical operators or quantifiers:*

$$af ::= t < t \mid t \leq t \mid t > t \mid t \geq t \mid t = t$$

*Additionally, an atomic formula is also a valid Presburger-Arithmetic formula.* ***Formulae*** *are defined the following way:*

$$f ::= af \mid \neg f \mid f \vee f \mid f \wedge f \mid \exists x f \mid \forall x f$$

*A Presburger-Arithmetic* ***sentence*** *is a formula without free variables.*

In Presburger's original theory, only the constants 0 and 1, the addition over the natural numbers and the predicate $\leq$ were used. It is entirely possible to restrict the definition of Presburger arithmetic to the structure $(\mathbb{N}, 0, 1, +, \leq)$ which has the same expressiveness as the definitions used here.

## 3 Algorithm

In the following we will present a procedure by Boudet and Comon [1] that involves constructing finite automata that can decide Presburger arithmetic. It is assumed that the reader is familiar with the notions of *deterministic finite automata* (DFA) and *non-deterministic finite automata* (NFA). For every Presburger formula we can construct an automaton that recognizes the set of its natural solution. Satisfiability of the formula then is reduced to emptiness checking of the automaton.

Given a Presburger formula, the solutions over the naturals for them can be viewed as a tuple $\vec{c} = (c_1, \ldots, c_n)$. For instance, a solution for $x + 2y - 3z \leq 2$ would be $(1, 2, 3)$. We encode the solution tuple in binary format with *least-significant bit first* (lsbf) encoding. The tuple $(1, 2, 3)$ is represented by $(10, 01, 11)$ or by its matrix representation $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$. All words of the automaton are interpreted as binary tuples in matrix representation. If a binary word is shorter than other numbers, zeroes "0" are added at the end until the same length is reached.

In the paper by Boudet and Comon [1] only procedures for constructing automata for inequalities $\sum_{i=1}^{n} a_i x_i \leq b$ and equations $\sum_{i=1}^{n} a_i x_i = b$ were presented. The rest of the formulae of Presburger arithmetic can be constructed with union, intersection, complementation and projection. In the following sections we describe the procedure for constructing an automaton that recognizes the natural solutions of an inequality.

### 3.1 Atomic Inequalities

Every inequality $t_1 \leq t_2$ can be rewritten to an inequality of the form $\sum_{i=1}^{n} a_i \cdot x_i \leq b$ where $a_i, b \in \mathbb{Z}$. The DFA $A_{\vec{a}\vec{x} \leq b}(Q, \Sigma, q_0, \delta, F)$ that recognizes the solution of an inequality $\sum_{i=1}^{n} a_i \cdot x_i \leq b$ (abbreviated with $\vec{a}\vec{x} \leq b$) has the following properties:

- The alphabet $\Sigma = \{0, 1\}^n$
- Every state is labeled after an inequality $\vec{a}\vec{x} \leq z$ with $z \in \mathbb{Z}$
  $\implies Q \subseteq \{\vec{a}\vec{x} \leq z \mid z \in \mathbb{Z}\}$
- The initial state $q_0$ is $\vec{a}\vec{x} \leq b$ (abbreviated with $b$). For convenience, every state is referred by its integer constant

In the subsequent sections, we will define the transition relation $\delta$ and the set of final states $F$ for the automaton.

**Transition Relation** Given an atomic inequality $\vec{a}\vec{x} \leq b$ with $n$ variables. Let $w = \sigma \cdot w_r$ be a word that encodes a tuple $\vec{c}$. Let $\vec{c_r}$ be the tuple that is encoded by $w_r$. Let $\vec{\sigma}$ be the tuple encoded by $\sigma$. Then $\vec{c}$ can be rewritten as $\vec{c} = 2 \cdot \vec{c_r} + \vec{\sigma}$ with the element-wise addition for tuples. It holds the following:

$$\vec{a} \cdot (2 \cdot \vec{c_r} + \vec{\sigma}) \leq b \iff \vec{a} \cdot \vec{c_r} \leq \frac{1}{2}(b - \vec{a} \cdot \vec{\sigma}) \tag{1}$$

That means the remainder $w_r$ has to encode a solution which satisfies the new inequality enforced by $\sigma$. This idea is used in the automaton construction by Boudet and Comon [1].

By convention, a state $q$ in the automaton accepts all solutions $w$ such that $\vec{a}\vec{c} \leq q$ if $w$ encodes a tuple $\vec{c}$. For an input $\sigma$, a successor state $q'$ of $q$ should therefore accept all words $w_r$ if $\sigma w_r$ is accepted by $q$. This can be achieved by labeling $q'$ with the inequality $\vec{a}\vec{x} \leq \frac{1}{2}(q - \vec{a}\vec{\sigma})$. If $\frac{1}{2}(q - \vec{a}\vec{\sigma})$ is not an integer, then the greatest integer $i < \frac{1}{2}(q - \vec{a}\vec{\sigma})$ is used since $\vec{a}\vec{x} \leq \frac{1}{2}(q - \vec{a}\vec{\sigma})$ and $\vec{a}\vec{x} \leq i$ have the same solutions in $\mathbb{N}^n$. The only case which leads to no solution is when $\vec{a}\vec{x} \leq b$ for all $a_i \geq 0$ and $b < 0$. This leads to the following definition:

**Definition 3 (Transition relation of the automaton for inequalities).** *Let $\vec{a}\vec{x} \leq q$ be a state of the automaton for an inequality $\vec{a}\vec{x} \leq b$. Let $\sigma \in \{0, 1\}^n$ where $n$ denotes the number of variables. Then the successor state $\vec{a}\vec{x} \leq q'$ is determined by the following relation:*

$$q' = \delta(q, \sigma) = \left\lfloor \frac{1}{2}(q - \vec{a} \cdot \vec{\sigma}) \right\rfloor$$

**Final States** Boudet and Comon consider only the states which accept the empty word $\varepsilon$ as final. The empty word encodes $\{0\}^n$. In this construction, these are the states that encode inequalities $\vec{a}\vec{x} \leq b$ with $b \geq 0$. The reason behind this is the following: Given a sequence $\vec{\delta}$ of transitions starting in $q$ and ending in $q'$. Assume that $q' < 0$. Thus, following from definition 3, the word $w$ composed by the letters of $\vec{\delta}$ encodes a tuple $\vec{c}$ such that $\vec{a}\vec{c} > q$. Therefore, $w$ would not be accepted by state $q$.

**Definition 4 (Accepting Condition of the automaton for inequalites).** *Given a state q from the automaton that accepts the solution for an inequality $\vec{a}\vec{x} \leq q$. State q is final if it accepts the empty word. Thus, every state with label $q \geq 0$ is final.*

Following from definition 3 and 4, the algorithm for constructing a DFA from an atomic inequality is shown in INEQtoDFA. To be completely precise, a trap state has to be also included in case of an inequality $\vec{a}\vec{x} \leq b$ with $\forall i.\ a_i \geq 0$ and $b < 0$. Especially for the case that the input formula is of the form. Then, the algorithm would return a single trap state. In the written algorithm INEQtoDFA the trap state has been omitted. The algorithm was presented in the Lecture *Automata and Formal Languages* held by Javier Esparza in 2019 at the *Technische Universität München* [9].

**Input:** an atomic inequality of the form $\sum_{i=1}^{n} a_i x_i \leq b$
**Result:** an automaton $A_{\vec{a}\vec{x}\leq b} = (Q, \Sigma, \delta, q_0, F)$
$Q \leftarrow \emptyset, \delta \leftarrow \emptyset, F \leftarrow \emptyset, q_0 \leftarrow b, W \leftarrow \{b\}, \Sigma \leftarrow \{0,1\}^n$;
**while** $W \neq \emptyset$ **do**
  pick $k$ from $W$, remove $k$ from $W$; add $k$ to $Q$;
  **if** $k \geq 0$ **then**
    add $k$ to $F$
  **for** $\sigma \in \Sigma$ **do**
    $j \leftarrow \left\lfloor \frac{1}{2}(k - \vec{a} \cdot \sigma) \right\rfloor$;
    **if** $j \notin Q$ **then**
      add $j$ to $W$
    add $(k, \sigma, j)$ to $\delta$;
**return** $(Q, \Sigma, \delta, q_0, F)$

**Procedure** INEQtoDFA

### 3.2 Termination

Let $\sum_{i=1}^{n} a_i x_i \leq b$ be an inequality. Let $s = \sum_{i=1}^{n} |a_i|$ be the sum of the absolute values of each coefficient. Boudet and Comon [1] stated that the construction presented in algorithm INEQtoDFA has at most $|b| + s$ states. Thus, the algorithm terminates. Furthermore, Esparza gave a more detailed proof in his lecture notes [9]. All states $q$ in the automaton satisfy the following property: It was proven that every state $q$ in the automaton satisfies the property $-|b| - s \leq q \leq |b| + s$. This can be proven through an induction on the states of the automaton. The initial state $b$ already satisfies the property. Let $q'$ be a successor state of another state in the automaton. Then for $\sigma \in \{0,1\}^n$ exists a state $q$ such that $q' = \lfloor \frac{1}{2}(q - \vec{a}\vec{\sigma}) \rfloor$. By assumption it holds for $q$ that $-|b| - s \leq q \leq |b| + s$. For $q'$, the same property holds:

$$-|b| - s \leq \frac{-|b| - 2s}{2} \leq \left\lfloor \frac{-|b| - s - \vec{a} \cdot \sigma}{2} \right\rfloor \leq q' \leq \left\lfloor \frac{|b| + s - \vec{a} \cdot \sigma}{2} \right\rfloor \leq |b| + s.$$

### 3.3 Extension to Formulae

We now have established a basis that can be extended to the rest of Presburger arithmetic. The procedure that extends to other formulae consists of using the complementation, intersection, union of regular languages as well as projection on a relation.

**Complementation** Let $A_f$ be an automaton for a formula $f$. If $A_f$ is a DFA, complementation can be performed in linear time by exchanging the set of final and non-final states which yields $A_{\neg f}$. In case of $A_f$ being an NFA, complementation becomes more costly with regard to size of the automaton. The NFA first has to be determinized which yields an equivalent DFA. This can be achieved by using the subset construction for NFAs [11]. For an NFA $A = (Q, \Sigma, q_0, F, \delta)$, the following description defines the equivalent DFA $A_D$:

- $Q_D \subseteq 2^Q$
- $\Sigma_D = \Sigma$
- $q_{D_0} = \{q_0\}$
- $F_D = \{q_D \in Q \mid \exists q \in q_D.\, q \in F\}$
- $\delta_D(q_D, i) = \bigcup_{q \in q_D} \delta(q, i)$

The determinization of an NFA has exponential time complexity $\mathcal{O}(2^n)$. After determinizing the NFA, complementation can be performed.

**Union and Intersection** Let $A_{f_1} = (Q_1, \Sigma_1, q_0, F_1, \delta_1)$ be an automaton recognizing the solutions of a formula $f_1$ and $A_{f_2} = (Q_2, \Sigma_2, p_0, F_2, \delta_2)$ recognizing the solutions of $f_2$. Again, automata theory provides us with a construction that can construct union and intersection of two automata [15]. For both union and intersection, the idea is to let both automata run at the same time. This construction is called the *product automaton*. In case of $\Sigma_1 = \Sigma_2$, the description of the product construction $A_{f_1 \circ f_2}$ for $A_{f_1}$ and $A_{f_2}$ is as follows:

- $Q \subseteq Q_1 \times Q_2$
- $q_{f_1 \circ f_2} = (q_0, p_0)$
- $\Sigma = \Sigma_1$
- For DFAs: $\delta_{f_1 \circ f_2}((q, p), i) = (q', p'), q' = \delta_1(q, i), p' = \delta_2(p, i)$
- For NFAs: $\forall q' \in \delta_1(q, i), \forall p' \in \delta_2(p, i)$: add $((q, p), i, (q', p'))$ to $\delta_{f_1 \circ f_2}$
- $(q, p) \in F \iff \begin{cases} q \in F_1 \wedge p \in F_2 \text{ for } A_{f_1 \wedge f_2} \\ q \in F_1 \vee p \in F_2 \text{ for } A_{f_1 \vee f_2} \end{cases}$

If $\Sigma_1 \neq \Sigma_2$, the automata $A'_{f_1} = (Q_1, \Sigma_1 \cup \Sigma_2, q_0, F_1, \delta_1)$ and $A'_{f_2} = (Q_2, \Sigma_1 \cup \Sigma_2, p_0, F_2, \delta'_2)$ are needed. To construct $A'_{f_i}, i \in \{1, 2\}$, simply add a trap state $q_t$ such that every transition with input $i \in (\Sigma_1 \cup \Sigma_2) \setminus \Sigma_i$ leads to $q_t$. The union/intersection of $A'_{f_1}$ and $A'_{f_2}$ have the same solutions as the union/intersection of $A_{f_1}$ and $A_{f_2}$.

Additionally, to construct the union for two NFAs, it suffices to take the union of both state sets, alphabets, initial states, transition relations and final states.

**Equations** A formula of the form $f_1 = f_2$ can be rewritten as $f_1 \leq f_2 \wedge f_2 \leq f_1$.

**Existential Quantification** Let $A_f$ be an automaton recognizing the solutions of a formula $f$. To construct the automaton $A_{\exists x_i.f}$ we perform a projection

$$\pi : \{x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n\} \to \{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$$

onto all variables of $f$ except $x_i$ [12]. In the automaton we simply omit the entry of the letter corresponding to $x_i$ for every transition in $A_f$. Afterwards, we might have to fix the set of final states where $A_{\exists x_i.f}$ does not accept another encoding of a correct solution.

For example, given an NFA as shown in Fig. 1 that accepts the tuple $(2, 4)$. If we

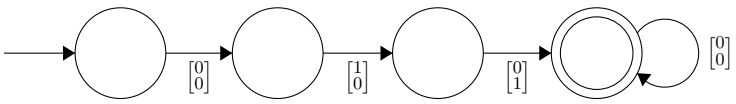


**Fig. 1.** NFA accepting $(2, 4)$

perform a projection on the first entry of the tuple, only "$0100^*$" would be accepted although "01" is also a valid encoding of 2. The third state in Fig. 1 has to be marked as final as well. In general, after performing a projection on the automaton, we also have to check for states that can reach final states through zeroes.

**Universal Quantifier** A universal formula $\forall x.\ f$ can be rewritten as $\neg\exists x.\ \neg f$.

### 3.4 Direct Construction: Equations

It is possible to directly construct an automaton that recognizes the natural solutions of an equation [1]. A similar idea used in the construction for inequalities is used. Let $\vec{a}\vec{x} = b$ be an equation and $w = \sigma \cdot w_r$ a possible solution of the equation. The natural tuple $\vec{c}$ is encoded by $w$ and $\vec{c_r}$ is encoded by $w_r$. As in the case of inequalities, a similar property holds:

$$\vec{a} \cdot (2 \cdot \vec{c_r} + \vec{\sigma}) = b \iff \vec{a} \cdot \vec{c_r} = \frac{1}{2}(b - \vec{a} \cdot \vec{\sigma}) \tag{2}$$

Equation (2) has a solution if and only if $b$ and $\vec{a}\vec{\sigma}$ have the same parity. For an input $\sigma$, if $(b - \vec{a}\vec{\sigma})$ is not even, the transition is directed to a trap state $s_t$. To summarize, the transition relation is defined as follows:

$$\delta(q, \sigma) = \begin{cases} \frac{1}{2}(q - \vec{a}\vec{\sigma}) & \text{if } q - \vec{a}\vec{\sigma} \textbf{ even} \\ s_t & \textbf{otherwise} \end{cases}$$

Procedure EQtoDFA shows the algorithm. Termination of INEQtoDFA is also guaranteed [1].

**Input:** an atomic equation of the form $\sum_{i=1}^{n} a_i x_i = b$
**Result:** a transducer $T = (Q, \Sigma, \delta, q_0, F)$

1 $Q \leftarrow \{s_t\}, \delta \leftarrow \emptyset, F \leftarrow \emptyset, q_0 \leftarrow b, W \leftarrow \{b\}, \Sigma \leftarrow \{0,1\}^n$;
2 **while** $W \neq \emptyset$ **do**
3  pick $k$ from $W$, remove $k$ from $W$; add $k$ to $Q$;
4  **if** $k = 0$ **then**
5  add $k$ to $F$
6  **for** $\sigma \in \Sigma$ **do**
7  $j \leftarrow \frac{1}{2}(k - \vec{a} \cdot \sigma)$;
8  **if** $k - \vec{a} \cdot \sigma$ *is even* **then**
9  add $(k, \sigma, j)$ to $\delta$;
10  **if** $j \notin Q$ **then**
11  add $j$ to $W$
12  **else**
13  add $(k, \sigma, s_t)$ to $\delta$
14 **return** $(Q, \Sigma, \delta, q_0, F)$

**Procedure** EQtoDFA

### 3.5 Direct Construction: Strict Inequalities

Lastly, it is also possible to directly construct the automaton for strict inequalities of the form $\vec{a}\vec{x} < b$. Let $w = \sigma \cdot w_r$ be a an accepted word. For a tuple $\vec{c_r}$ of naturals encoded by $w_r$ a similar property as in the case for inequalities holds:

$$\vec{a} \cdot (2 \cdot \vec{c_r} + \vec{\sigma}) < b \iff \vec{a}\vec{c_r} < \frac{1}{2}(b - \vec{a}\vec{\sigma}) \tag{3}$$

If $\frac{1}{2}(b - \vec{a}\vec{\sigma}) \in \mathbb{N}$, then we can add a transition $(b, \sigma, \frac{1}{2}(b - \vec{a}\vec{\sigma}))$ to the transition relation of the automaton. Otherwise, we use $\lceil \frac{1}{2}(b - \vec{a}\vec{\sigma}) \rceil$, since $\vec{a}\vec{x} < \frac{1}{2}(b - \vec{a}\vec{\sigma})$ has the same solutions in $\mathbb{N}^n$ as $\vec{a}\vec{x} < \lceil \frac{1}{2}(b - \vec{a}\vec{\sigma}) \rceil$. The transition relation for the automaton is defined as follows:

$$\delta(q, \sigma) = \left\lceil \frac{1}{2}(q - \vec{a}\vec{\sigma}) \right\rceil, \sigma \in \{0,1\}$$

The only lines that have to be modified in the original INEQtoDFA algorithm are line 4 and 7. We would mark state $k$ as final only if $k > 0$ and instead of using the floor function we use the ceiling function.

## 4 Example

We proceed to construct an automaton that accepts the set of solutions for $\exists x.\ 2x - y = 0$. Table 1 shows the execution steps for the construction process. Note that the transitions to the trap state have been omitted for the sake of compactness. Fig. 2 shows the DFA for $2x - y = 0$. The trap state has been omitted here. In order to get the automaton for $\exists x.\ 2x - y = 0$ we omit in each transition the first entry of the vector. Afterwards, we do not have to fix the set of finals states since state 1 will not reach state 0 with input "0". Fig. 3 shows the NFA for $\exists x.\ 2x - y = 0$. Fig. 4 shows

the equivalent DFA for the NFA in Fig. 3. The transition to the empty state from state 0 has been omitted as well.

**Table 1.** Execution steps of EQtoDFA for $2x - y = 0$

| i | W | Q | New Transitions |
|---|---|---|---|
| 0 | $\{0\}$ | $\emptyset$ | - |
| 1 | $\{-1\}$ | $\{0\}$ | $(0, \begin{bmatrix}0\\0\end{bmatrix}, 0), (0, \begin{bmatrix}1\\0\end{bmatrix}, -1)$ |
| 2 | $\emptyset$ | $\{0, -1\}$ | $(-1, \begin{bmatrix}1\\1\end{bmatrix}, -1), (-1, \begin{bmatrix}0\\1\end{bmatrix}, 0)$ |

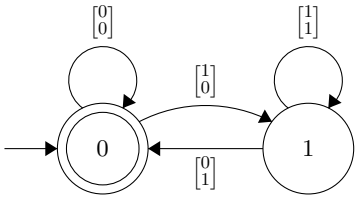


**Fig. 2.** DFA for $2x - y = 0$

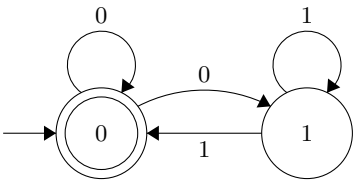


**Fig. 3.** NFA for $\exists x.2x - y = 0$

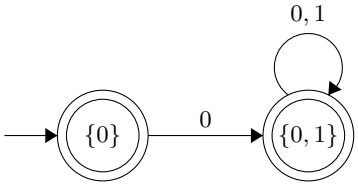


**Fig. 4.** DFA for $\exists x.2x - y = 0$

For the emptiness check, it would have been sufficient to perform a check on the NFA in Fig. 3 and avoid the determinization step. We would have seen that the automaton is not empty, therefore the formula is satisfiable. The solutions of the formula are the words formed by the expression $0\{0, 1\}^*$ as seen in Fig. 4 which are all binary encodings of even numbers.

## 5 Further Improvements

### 5.1 Systems of Linear Elementary Formulae

Given a system of linear equations:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ &\vdots \\ a_{k,1}x_1 + a_{k,2}x_2 + \cdots + a_{k,n}x_n &= b_k \end{aligned}$$

The question is whether a solution $\vec{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{N}^n$ exists. Intersecting the automata $A_1, A_2, \ldots, A_k$ for $\sum_{i=0}^{n} a_{1,i}x_i = b_1$, $\sum_{i=0}^{n} a_{2,i}x_i = b_2, \ldots, \sum_{i=0}^{n} a_{k,i}x_i = b_k$ respectively returns an automaton that recognizes the solutions for this system. However, Boudet and Comon [1] showed a procedure that directly constructs an automaton for this system.

The system can be rewritten to the following concise form:

$$\bigwedge_{i=1}^{k} \sum_{j=1}^{n} a_{i,j} \cdot x_j = b_i. \tag{4}$$

Each state is labeled after a conjunction of linear equations. The initial state is (4) or simply $\vec{b} = (b_1, b_2, \ldots, b_k)$. To compute the successor states, the same notion for constructing the automaton for equations is applied. For an input $\vec{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, the following holds:

$$\bigwedge_{i=1}^{k} \sum_{j=1}^{n} a_{i,j}(2c'_j + \sigma_j) = b_i \iff \bigwedge_{i=1}^{k} \sum_{j=1}^{n} a_{i,j}c'_j = \frac{1}{2}\left(b_i - \sum_{j=1}^{n} a_{i,j}\sigma_j\right) \tag{5}$$

The successor state then is determined as described in the left-hand side of (5) if $b_i - \sum_{j=1}^{n} a_{i,j}\sigma_j$ is even for all $i$. The final state is the system where $k_i = 0$ for all $i$.

### 5.2 Extension to Integer Solutions

The algorithm can be extended to the solutions over the integers using NFAs and *two's-complement encoding*. We also use lsbf representation again. Stacking the same sign bit does not alter the value of the bit sequence. For example, $011$ encodes $0+2-4=-2$. So does $0111$ with $0+2+4-8=-2$. Another example is $\left[\begin{smallmatrix}0&1&1\\1&1&0\end{smallmatrix}\right]$ where the sign bit tuple is $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$. The expression $\left[\begin{smallmatrix}0&1&1\\1&1&0\end{smallmatrix}\right]\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]^*$ therefore encodes the same tuple.

Let $w = \sigma \cdot w_r$ be a binary word. Let $\vec{c}$ be an integer tuple encoded by $w$ and $\vec{c_r}$ encoded by $w_r$. For $w_r$ there are two cases:

- Case $w_r \neq \varepsilon$: It holds that $\vec{c} = (2 \cdot \vec{c_r} + \sigma)$
- Case $w_r = \varepsilon$: $w = \sigma$ is a sign bit

A single bit in two's-complement encodes either $0$ or $-1$. Thus, if the sub-word $w_r$ is empty, then $w = \sigma$ encodes $-\vec{\sigma}$. In the extended construction, we always have to check for these two cases in every state. Let $q$ be a state of an automaton for an inequality $\vec{a}\vec{x} \leq b$. Let $w = \sigma \cdot w_r$ be a word. For the case of $w_r \neq \varepsilon$, we calculate the successor state as defined for inequalities. In case of $w_r = \varepsilon$, we check whether $\vec{a}(-\vec{\sigma}) \leq q$ holds. If the check is successful, then an additional transition is added which leads to a new final state $q_f$. The automaton that recognizes the integer solutions of a formula has only one final state which is $q_f$.

For inequalities, strict inequalities and equations, the transition relation of their respective transition relation is extended to the following definition:

$$(\vec{a}\vec{x} \leq b) :\ \delta(q,\sigma) = \begin{cases} \left\{\left\lfloor \frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right)\right\rfloor, q_f\right\} & q + \vec{a}\vec{\sigma} \geq 0 \\ \left\{\left\lfloor \frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right)\right\rfloor\right\} & \textbf{otherwise} \end{cases}$$

$$(\vec{a}\vec{x} < b) :\ \delta(q,\sigma) = \begin{cases} \left\{\left\lceil \frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right)\right\rceil, q_f\right\} & q + \vec{a}\vec{\sigma} > 0 \\ \left\{\left\lceil \frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right)\right\rceil\right\} & \textbf{otherwise} \end{cases}$$

$$(\vec{a}\vec{x} = b) :\ \delta(q,\sigma) = \begin{cases} \left\{\frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right), q_f\right\} & q - \vec{a}\vec{\sigma} \textbf{ even} \wedge q + \vec{a}\vec{\sigma} = 0 \\ \left\{\frac{1}{2}\left(q - \vec{a}\vec{\sigma}\right)\right\} & q - \vec{a}\vec{\sigma} \textbf{ even} \end{cases}$$

### 5.3 Divisibility Relation

Lastly, it is also possible to construct the automaton for a divisibility relation $d|t + c$ where $d \geq 2$ and $t$ denotes a term and $c \in \mathbb{Z}$ a constant. A formula $d|t + c$ denotes all solutions $\vec{s}$ such that $t\big[\vec{s}\big] + c$ is divisible by $d$. The construction used here was presented in a paper by Klaedtke [13]. The automaton uses *most-significant-bit-first* format with two's-complement encoding. Let $r(a, b)$ be a function that calculates the remainder of $a$ divided by $b$. The automaton $A_{d|t+c} = (Q, \Sigma, q_0, \delta, F)$ is structured as follows:

- $Q = \{q_0, 0, 1, \ldots, d-1\}$
- $\Sigma = \{0, 1\}^n$
- $\delta(q, \sigma) = \begin{cases} r(-t\big[\vec{\sigma}\big], d), & \text{if } q = q_0 \\ r(2q + t\big[\vec{\sigma}\big], d), & \text{otherwise} \end{cases}$
- $q \in F \iff q \in Q \cap \mathbb{N} \wedge d|q + c$

## 6 Complexity

Regarding the size of the automaton construction by Boudet and Comon [1], a distinction between the quantifier-free fragment and full logic can be made. For an atomic formula $f$, let $K(f) = \sum_{i=1}^{n} |a_i| + |b|$ be the sum of the absolute values of all coefficients $a_i$ and the absolute value of the constant. Let $V(f)$ be the number of variables. Then, the state complexity of an automaton for a quantifier-free Presburger formula is $\mathcal{O}(2^n)$ with $n = K(f) + V(f)$. For the full logic it is $\mathcal{O}(2^{2^{2^n}})$.

Klaedtke [13] came to a similar conclusion regarding the size of automata for Presburger arithmetic. He showed that the size of the minimal DFA for a Presburger arithmetic formula is triple exponential to the length of the formula. This upper bound is only achievable when the automaton is minimized after a costly operation such as negation.

## 7 Final Words

The automata-theoretic approach for deciding Presburger arithmetic not an optimized one. The constructed automaton is not necessarily minimal which means that further minimization steps need to be considered. And regarding the quantifier-free fragment of Presburger arithmetic, modern integer programming solvers outperform the automata-based method [10].

Deciding Presburger arithmetic is a rewarding task since insights gained research directly benefit integer programming problems as well. Solvers based on quantifier elimination can be found in the Isabelle proof assistant [6, 17]. And research in other related fields indirectly or even directly benefit research on Presburger arithmetic as shown in [10] or in [20]. Research using Presburger arithmetic is commonly found in the area of Model Checking. Some examples using Presburger arithmetic are found in [5, 7, 2, 8].

## References


[1] Alexandre Boudet and Hubert Comon. "Diophantine Equations, Presburger Arithmetic and Finite Automata". In: *Trees in Algebra and Programming - CAAP'96, 21st International Colloquium, Linköping, Sweden, April, 22-24, 1996, Proceedings*. Ed. by Hélène Kirchner. Vol. 1059. Lecture Notes in Computer Science. Springer, 1996, pp. 30–43.

[2] Véronique Bruyère, Emmanuel Dall'Olio, and Jean-FranÇois Raskin. "Durations, Parametric Model-Checking in Timed Automata with Presburger Arithmetic". In: *STACS 2003*. Ed. by Helmut Alt and Michel Habib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 687–698.

[3] Julius Richard Büchi. "On a Decision Method in Restricted Second Order Arithmetic". In: *Proc. of the International Congress on Logic, Method and Philosophy of Science, 1962*. Stanford University Press. 1962, pp. 1–11.

[4] Julius Richard Büchi. "Weak Second-Order Arithmetic and Finite Automata". In: *Mathematical Logic Quarterly* 6.1-6 (1960), pp. 66–92.

[5] Tevfik Bultan, Richard Gerber, and William Pugh. "Symbolic Model Checking of Infinite State Systems Using Presburger Arithmetic". In: *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel, June 22-25, 1997, Proceedings*. Ed. by Orna Grumberg. Vol. 1254. Lecture Notes in Computer Science. Springer, 1997, pp. 400–411.

[6] Amine Chaieb and Tobias Nipkow. "Verifying and Reflecting Quantifier Elimination for Presburger Arithmetic". In: *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*. Ed. by Geoff Sutcliffe and Andrei Voronkov. Vol. 3835. Lecture Notes in Computer Science. Springer, 2005, pp. 367–380.

[7] Hubert Comon and Yan Jurski. "Multiple Counters Automata, Safety Analysis and Presburger Arithmetic". In: *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*. Ed. by Alan J. Hu and Moshe Y. Vardi. Vol. 1427. Lecture Notes in Computer Science. Springer, 1998, pp. 268–279.

[8] Stéphane Demri et al. "Model-checking CTL* over flat Presburger counter systems". In: *Journal of Applied Non-Classical Logics* 20.4 (2010), pp. 313–344.

[9] Javier Esparza. "Automata and Formal Languages". University Lecture Notes, Technische Universitat München. 2019.

[10] Vijay Ganesh, Sergey Berezin, and David L. Dill. "Deciding Presburger Arithmetic by Model Checking and Comparisons with Other Methods". In: *Formal Methods in Computer-Aided Design, 4th International Conference, FMCAD 2002, Portland, OR, USA, November 6-8, 2002, Proceedings*. Ed. by Mark D. Aagaard and John W. O'Leary. Vol. 2517. Lecture Notes in Computer Science. Springer, 2002, pp. 171–186.

[11] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. 3rd Edition. Pearson international edition. Addison-Wesley, 2007.

[12] Galina Jirásková and Tomás Masopust. "State Complexity of Projected Languages". In: *Descriptional Complexity of Formal Systems - 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25-27, 2011. Proceedings*. Ed. by

Markus Holzer, Martin Kutrib, and Giovanni Pighizzini. Vol. 6808. Lecture Notes in Computer Science. Springer, 2011, pp. 198–211.

[13] Felix Klaedtke. "On the Automata Size for Presburger Arithmetic". In: *19th IEEE Symposium on Logic in Computer Science (LICS), 14-17 July 2004, Turku, Finland, Proceedings*. IEEE Computer Society, 2004, pp. 110–119.

[14] Hugues Marchand et al. "Cutting planes in integer and mixed integer programming". In: *Discrete Applied Mathematics* 123.1 (2002), pp. 397–446.

[15] John C Martin. *Introduction to Languages and the Theory of Computation*. Vol. 4. McGraw-Hill New York, 1991.

[16] Jiří Matoušek and Bernd Gärtner. "Integer Programming and LP Relaxation". In: *Understanding and Using Linear Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 29–40.

[17] Tobias Nipkow. "Linear Quantifier Elimination". In: *Journal of Automated Reasoning* 45.2 (2010), pp. 189–212.

[18] Martin Pincus. "Letter to the Editor—A Monte Carlo Method for the Approximate Solution of Certain Types of Constrained Optimization Problems". In: *Operations Research* 18.6 (1970), pp. 1225–1228.

[19] Mojżesz Presburger. "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt". In: *Comptes Rendus du I congres de Mathématiciens des Pays Slaves. Slaves, Warsaw* (1929), pp. 92–101.

[20] William Pugh. "The Omega Test: A Fast and Practical Integer Programming Algorithm for Dependence Analysis". In: *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*. Supercomputing '91. Albuquerque, New Mexico, USA: Association for Computing Machinery, 1991, pp. 4–13.

[21] Stuart Jonathan Russell and Peter Norvig. *Artificial intelligence - a modern approach, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, 2003.