# Diophantine Equations, Presburger Arithmetic and Finite Automata*

Alexandre Boudet and Hubert Comon

LRI, CNRS URA 410
Bât 490, Université Paris-Sud, Centre d'Orsay
91405 Orsay Cedex, France

**Abstract.** We show that the use of finite automata provides a decision procedure for Presburger Arithmetic with optimal worst case complexity.

## Introduction

Solving linear equations and inequations with integer coefficients in the set Nof non-negative integer plays an important role in many areas of computer science, such as associative commutative unification, constraint logic programming, compiler optimization,... The first-order theory of Nwith addition 0 and 1 is known as *Presburger arithmetic* and has been shown decidable as early as in 1929 [5]. The special case of linear Diophantine equations has been studied even earlier [2]. Much work has been devoted recently to improve the effectiveness of known methods, as well as in designing new efficient algorithms [3, 1, ?, ?]. For example, E, Domenjoud and A.-P. Tomas in [?] study old methods of Elliot and Mac Mahon [2, 4], improving their algorithms and extending so as to be able to solve more complex systems including inequations ($\geq$) and disequations ($\neq$).

In this paper, we follow a similar approach: we revisit Büchi's technique [?] in the context of Diophantine equations systems and their extension up to Presburger arithmetic.

The most famous result of Büchi is the decidability of the sequential calculus: the second-order monadic logic with one successor (S1S). It is out of the scope of this paper to recall all the background of this result, which we do not need in its full generality. Let us just recall that, in the case of the *weak* secon-order monadic logic WS1S (when the set variables range over finite sets only), Büchi's result can be restated as': "a subset of the free monoid $\{a_1, \ldots, a_n\}^*$ is recognizable by a finite state automaton if and only if it is definable in WS1S". This result is relevant to Presburger arithmetic; to see it, we proceed as follows. Natural numbers can be seen as finite words over the alphabet $\{0, 1\}$: it is sufficient to consider their binary representation, which we write from right to left. The representation is not unique, as we may add some zeros on the right. For example, the number thirteen can be represented as $1011, 10110, \ldots$. More generally, tuples of natural numbers can be represented in binary notation as words over the

* This research was supported in part by the HCM Network SOL.

alphabet $\{0,1\}^n$, simply by stacking them, using an equal length represnetation for each number. For example, the pair (thirteen,four) can be represented as $\substack{1011\\0010}$ (or $\substack{10110\\00100}$...). Now, there is a finite automaton which accepts the tripes $(x, y, z)$ of natural numbers such that $x = y + z$: a two state automaton (one (final) state for "no-carry"and one state for "carry") is actually sufficient. Hence, by Büchi's theorem, this set of words is definable in WS1S. Now, we may use arbitrary logical connectives as well as quantifications over finite sets (which turn out to correspond to quantification over natural numbers), we stay within WS1S, which is decidable: Presburger arithmetic is now embedded in WS1S.

From this observation, finite automata give a possible device for solving linear Diophantine problems. How efficient is the method ? What is its practical relevance ? Which problems can it solve ? That are some of the questions we aim at answering in this paper.

To compare with previous methods, we should first emphasize some weaknesses and some strengths of the automata approach:

**Strenghts** . First, the algorithm itself is extremely simple (it can be implemented in less than two hours). Adding disequations, disjunctions, inequations is straightforward and does not increase the complexity. It is also easy to add quantifiers, to the price of an increased complexity. Similarly, it is possible to add any recognizable predicate over natural number (for example the predicate "to be a power of 2"), while keeping decidability.

**Weaknesses** Usually, in linear Diophantine equation solving, the outcome is a *basis* of the set of solutions, or some *parametric representation* [1, ?, ?]. The outcome of our technique is a finite state automaton which recognizes the set of solutions (and whose emptiness is known). Extracting a basis of solutions from the automaton might be a complex step. Hence our approach cannot be used for e.g. associative-commutative unification which requires a particular representation of the set of solutions.

One of the major issue is efficiency. On the theoretical side, one contribution of this paper is to show that, as a decision technique, our algorithm is near to be optimal. More precisely, we show that our algorithm runs in exponential time for the existential fragment of Presburger arithmetic, which is known to be NP-complete. It runs in triply exponential time for the whole Presburger arithmetic, which is known to be complete for double exponential space [?]. On the practical side, we carried several experiments with a prototype implementation, which show that our methode is competitive[2] (for the decision problem).

The paper is organized as follows. In section 1 we present the general method for a single equation and explain why it is possible to derive a (naive) decision procedure for Presburger arithmetic. In section **??** we consider the existential fragment and address implementation issues and complexity. In section **??** we

[2] We do not report of computation-times here, compared with other methods since they would be meaningless; our computation times often show that our method is more efficient, however other algorithms are in general not only dedicated to the decision problems.

consider the full Presburger arithmetic and show the complexity of our algorithm.

# 1 Diophantine equations and finite automata

## 1.1 Recognizability of the solutions of a linear Diophantine equation

We show here how to encode tuples of naturals as words on a given alphabet in such a way that the set of solutions of a linear Diophantine equation is a recognized by a finite state automaton.

Rather than showing formally the construction, we sketch it and develop an example. It should be clear from the example how to compute the automaton in the general case

Consider the linear Diophantine equation

$$(e) \qquad\qquad x + 2y - 3z = 2$$

Equation $(e)$ has to be satisfied modulo 2: if $(c_1, c_2, c_3)$ is a solution of $e$, the respective reminders $b_1, b_2, b_3$ of $c_1, c_2, c_3$ are solution of (e) modulo 2, i.e. $b_1 = b_3$ in our case. Let $S_2(e)$ be the set of solutions of $e$ modulo 2. Here $S_2(e) = \{(0,0,0), (0,1,0), (1,0,1), (1,1,1)\}$. Now, for each triple $(b_1, b_2, b_3) \in S_2(e)$,

$$c_1 - b_1 + 2(c_2 - b_2) - 3(c_3 - b_3) = 2 - (b_1 + 2b_2 - 3b_3)$$

which can be divided by 2: the quotients $c'_1, c'_2, c'_3$ of $c_1, c_2, c_3$ by 2 respectively have to satisfy the new equation:

$$(e(b_1, b_2, b_3)) \qquad x + 2y - 3z = \frac{2 - (b_1 + 2b_2 - 3b_3)}{2}$$

We have now split $(e)$ into an equivalent disjunction of 4 new equations on the quotients by 2 of $x, y, z$.

Let us express this step in formal languages. A *coding* of a non-negative integer $c$ is a word $c_0 \ldots c_m$ such that each $c_i$ is 0 or 1 and $c = \sum_{i=1}^{m} c_i 2^i$. In other words, we consider the binary representation of $c$, written from right to left, possibly completed by zeros on the right. A tuple $(c_1, \ldots, c_n)$ of non-negative integers will be (ambiguously) encoded by stacking any representations of $c_1, \ldots, c_n$ which have the same length. For example, the triple $(3, 1, 1)$ can be coded as $\substack{11\\10\\10}$. In this way, any $n$-uple of non-negative integers can be seen as a word over the alphabet $\Sigma = \{0,1\}^n$. Let $[\![e]\!]$ be the set of words (the language) which are solutions of $e$. The above decomposition shows that

$$[\![e]\!] = \bigcup_{(b_1,b_2,b_3)\in S_2(e)} \substack{b_1\\b_2\\b_3} \cdot [\![e(b_1, b_2, b_3)]\!]$$

Which can be developed in our example:

$$\begin{aligned}\llbracket x+2y-3z=2\rrbracket = \; & \begin{smallmatrix}0\\0\\0\end{smallmatrix} \cdot \llbracket x+2y-3z=1\rrbracket \cup \begin{smallmatrix}0\\1\\0\end{smallmatrix} \cdot \llbracket x+2y-3z=0\rrbracket \\ & \cup \begin{smallmatrix}1\\0\\1\end{smallmatrix} \cdot \llbracket x+2y-3z=2\rrbracket \cup \begin{smallmatrix}1\\1\\1\end{smallmatrix} \cdot \llbracket x+2y-3z=1\rrbracket\end{aligned}$$

Now, we can derive similar equations for the new equations which appear in the right member above, for example:

$$\begin{aligned}\llbracket x+2y-3z=1\rrbracket = \; & \begin{smallmatrix}0\\0\\1\end{smallmatrix} \cdot \llbracket x+2y-3z=2\rrbracket \cup \begin{smallmatrix}0\\1\\1\end{smallmatrix} \cdot \llbracket x+2y-3z=1\rrbracket \\ & \cup \begin{smallmatrix}1\\0\\0\end{smallmatrix} \cdot \llbracket x+2y-3z=0\rrbracket \cup \begin{smallmatrix}1\\1\\0\end{smallmatrix} \cdot \llbracket x+2y-3z=-1\rrbracket\end{aligned}$$

Assuming (what will be proved below) that this process terminates, we get a system of left-linear equations over $\Sigma^*$, whose solution is then a regular language; as a final outcome, we get the automaton of figure 1.

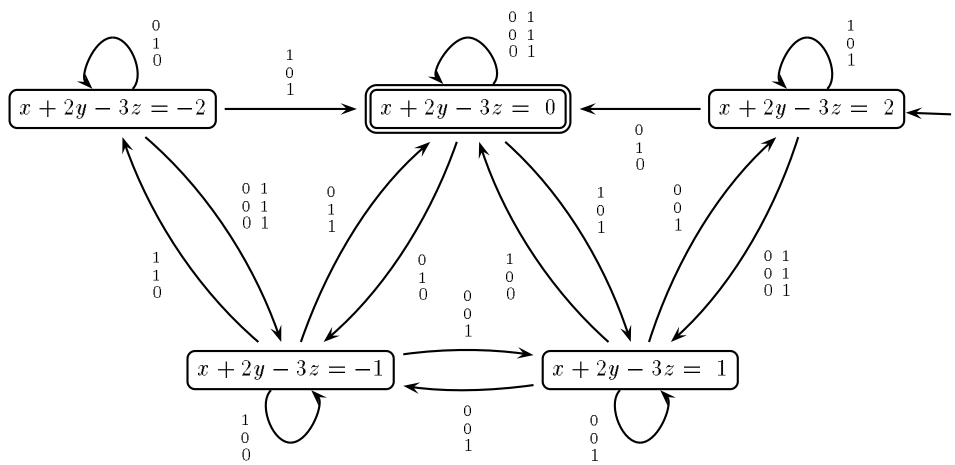


**Fig. 1.** The automaton which recognizes the solutions of $x+2y-3z=2$

The initial state is $\llbracket e\rrbracket$ and the final state is $\llbracket x+2y-3z=0\rrbracket$: every word in the set of solution might be followed eventually by a sequence of $\begin{smallmatrix}0\\0\\0\end{smallmatrix}$.

More generally, the automatin is constructed as follows: we start from a set of states containing $[e]$ where $e$ is the equation to be soved and the junk state $[-]$. The transition rules $T$ only contain initially the transition from $[-]$ to itself by any letter of the alphabet $\Sigma$. Then we saturate $Q$ and $T$ according to the rules:

If $[a_1x_1 + \cdots + a_nx_n = k] \in Q$ and $\alpha = \begin{matrix} b_1 \\ \vdots \\ b_n \end{matrix} \in \Sigma$ then

**If** $k - (a_1b_1 + \cdots + a_nb_n)$ **is even then**

$$\begin{cases} [a_1x_1 + \cdots + a_nx_n = k] \xrightarrow{\alpha} [a_1x_1 + \cdots + a_nx_n = \frac{k-(b_1a_1+\cdots+b_na_n)}{2}] \\ \qquad \text{is added to } T \\ [a_1x_1 + \cdots + a_nx_n = \frac{k-(b_1a_1+\cdots+b_na_n)}{2}] \\ \qquad \text{is added to } Q \end{cases}$$

**If** $k - (a_1b_1 + \cdots + a_nb_n)$ **is odd then**

$$[a_1x_1 + \cdots + a_nx_n = k] \xrightarrow{\alpha} [-] \text{ is added to } T$$

The only initial state is $[e]$ and the only final state is $[a_1x_1 + \cdots a_nx_n = 0]$.

We are now left to show that the set of states (equations) that can be reached from any initial state (equation) $[a_1x_1 + \cdots + a_nx_n = k]$ is finite. This is so, because if $|k| > \Sigma_{i=1}^{n}|a_i|$, then for any letter $\alpha = b_1 \cdots b_n$, $|k| > |\frac{k-(b_1a_1+\cdots+b_na_n)}{2}|$. Hence, for an equation $e$ of the form $a_1x_1 + \cdots + a_nx_n = k$ the number of states of the automaton is bounded by $|k| + \Sigma_{i=1}^{n}|a_i|$.

**Proposition 1.** *Let $e$ be the linear Diophantine equation $a_1x_1 + \cdots + a_nx_n = k$. The set of solutions of $e$ is recognized by a finite, complete and deterministic automaton $A$ which has at most $|k| + \Sigma_{i=1}^{n}|a_i|$ states and at most $2^n$ transitions from any state. If the* size *of $e$ is the sum $|e|$ of the lengths of $a_1, \ldots, a_n, k$, written in binary plus the number $n$ of variables, then the automaton $A$ can be built in time $2^{|e|}$.*

Remark: $A$ is not necessarily minimal: for instance if there is a state $[e]$ with unsatisfiable $e$, then this state should be identified with $[-]$. But two states $[e]$ and $[e']$ from which there is a path leading to the accepting state with different $e$ and $e'$ will never be identified in the minimal deterministic automaton. Indeed, $e$ and $e'$ are of the form $a_1x_1+\cdots+a_nx_n = k$ and $a_1x_1+\cdots+a_nx_n = k'$, they differ only by the constant of the right-hand side, hence, they cannot have the same sets of solutions. Hence, the minimalization of the automaton just consists in our case of identifying all the states from which the accepting state is inaccessible with $[-]$. Note that this operation is can be performed in linear time wrt the size of $A$.

We introduce a convenient notation that will be used in the remainder of the paper.

*Notation 2.* Let $\phi$ be a formula of Presburger arithmetic with variables $x_1, \ldots, x_n$. Let $\alpha = \begin{matrix} b_1 \\ \vdots \\ b_n \end{matrix} \in \{0,1\}^n$.

We denote by $\phi \otimes \alpha$ the formula $\phi\{x_1 \mapsto 2x_1 + b_1, \ldots, x_n \mapsto 2x_n + b_n\}$.

Actually, the state reached from a state labeled with formula $\phi$ reading $\alpha$ is labeled by a formula equivalent to $\phi \otimes \alpha$.

### 1.2 Decidability of Presburger Arithmetic

Büchi in the early sixties proved the following result:

> A subset of $(\{0,1\}^n)^*$ is recognizable iff it is definable in WS1S (the weak second-order monadic logic with one successor).

We do not want to recall here all the background on the sequential calculus. We refer e.g. to [7] for more details. Let us simply describe the resulting algorithm for Presburger arithmetic decision, hereafter called "the automaton algorithm":

We assume given, for each atomic formula $\phi$ an automaton $\mathcal{A}_\phi$ which accepts the solutions of $\phi$. Then, for every formula $\psi$, the automaton accepting $\psi$ is inductively defined as follows:

- $\mathcal{A}_{\phi_1 \wedge \phi_2} = \mathcal{A}_{\phi_1} \cap \mathcal{A}_{\phi_2}$, i.e. the automaton accepting the intersection language. It is computed in quadratic time by standard means.
- $\mathcal{A}_{\phi_1 \vee \phi_2} = \mathcal{A}_{\phi_1} \cup \mathcal{A}_{\phi_2}$, i.e. the automaton accepting the union language. It is computed in linear time by standard means.
- $\mathcal{A}_{\neg\phi}$ is the automaton accepting the complement of the language accepted by $\mathcal{A}_\phi$. Its computation may require a determinization, which can yield an exponential blowup in the worst case.
- $\mathcal{A}_{\exists x.\phi}$ is computed by *projection*, another standard operation (see e.g. [7]) requiring linear time.
- $\mathcal{A}_{\forall x.\phi} = \mathcal{A}_{\neg\exists x.\neg\phi}$. This translation can be quite expensive: in principle the resulting automaton could be doubly exponential w.r.t. the original one. Actually, it is not hard to see that this step is simply exponential in the worst case[3]

Finally, if one wants to decide the validity of a sentence $\phi$, it is sufficient to compute $\mathcal{A}_\phi$ and check whether it contains an accessible final state.

## 2 Unquantified formulae

We first show how to compute directly an deterministic automaton which accepts the solutions of disequations and inequations. Then we show how to compute efficiently a deterministic automaton for conjunctions (systems) of such atomic formulae.

[3] We don't have any reference at hand. Anyway, this does not play any role in the following.

### 2.1 Disequations and inequations

**Disequations** Turning the automaton $A$ recognizing the solutions of an equation $e$ into one recognizing the solutions of the disequation $\neg e$ is straightforward: $A$ being complete and deterministic, one just has to say that the accepting state is no longer accepting and that all the other states are accepting.

**Inequations** Since the language of the solutions of an equation $e$ is recognizable, automata theory gives us a straightforward solution for inequations. Indeed, the solutions of the inequation $s \leq t$ are the same as those of the formula $\exists x s = t + x$. But we can directly build an automaton for recognizing the solutions of an inequation. Again, a state of the automaton will be labeled by a formula (inequation), and the language recognized starting from a given state $[i]$ encodes the language of the solutions of the inequation $i$. Let us write an inequation under the form:

$$(i) \quad a_1 x_1 + \cdots + a_n x_n \leq k$$

where the $a_1, \ldots, a_n, k \in Z$. The only case where this inequation has no solutions is when $k$ is negative and $a_1, \ldots, a_n$ are all positive. In this case, the automaton is a trivial one with only one non-accepting state. Otherwise, similarily as for equations, the word $\alpha \bullet w$ encodes solution of $i$ if and only if $w$ encodes a solution of $i \otimes \alpha$. The transitions scheme is similar to that for equations, except that the two sides of an inequation need not have the same parity. Consider the inequation (state) $[x + 2y - 3z \leq 1]$. If the letter 101 is read, (meaning that $x$ and $z$ are odd and $y$ even), the remainder of the word must encode a solution of the inequation $2x+1+4y-6z-3 \leq 1$ which is equivalent to $2(x+2y-3z) \leq 3$ or $x+2y-3z \leq \frac{3}{2}$. But this latter inequation has the same solutions in $N^3$ than $x + 2y - 3z \leq 1$. Hence the transition schemata for inequations ($\lfloor \cdot \rfloor$ denotes the integer part of a rational number):

**Transition**

$$[a_1 x_1 + \cdots + a_n x_n \leq k] \xrightarrow{\alpha} [a_1 x_1 + \cdots + a_n x_n \leq \lfloor \frac{k - (b_1 a_1 + \cdots + b_n a_n)}{2} \rfloor]$$

$$\text{if } \alpha = b_1 b_2 \cdots b_n \text{ and } k \geq 0 \text{ or } a_i < 0 \text{ for some } i.$$

**Fail**

$$[a_1 x_1 + \cdots + a_n x_n \leq k] \xrightarrow{\alpha} [-]$$

$$\text{if } k < 0 \text{ and } a_1 > 0, \ldots, a_n > 0.$$

The termination argument is the same as in the (dis)equation case. The accepting states are all those labeled by an inequation for which $(0, \ldots, 0)$ (encoded by $\epsilon$) is a solution, that is the states $[a_1 x_1 + \cdots + a_n x_n \leq k]$ for which $k \geq 0$. The automaton computed using the above transition schemes is deterministic and complete, but it is not minimal: consider the inequation $2x - 2y \leq 1$. It has the same solutions as $x \leq y$ in $N^2$. The minimal automaton recognizing the solutions of $x \leq y$ has two states, but the reader can check that our method will generate a larger automaton. Remember though that minimalisation can be performed in polynomial time.

### 2.2 Unquantified formulae

As always, automata theory gtives us a straightforward procedure. An automaton recognizing the solutions of a conjunction of atomic formulae $\phi_1 \wedge \cdots \wedge \phi_m$ is the product of the automata $A_1, \ldots, A_n$ recognizing the solutions of $\phi_1, \ldots, \phi_n$ respectively. But in the case of equations, we have seen that in state $[e]$, reading letter $\alpha$ a state different than $[-]$ is reached if and only if $e \otimes \alpha$ has its left-hand and right-hand sides of the same parity. In case the equation $a_1 x_1 + \cdots + a_n x_n = k$ has at least one odd coefficient $a_i$, there will be exactly $2^{n-1}$ transitions from any state to a state different than $-$ and $2^{n-1}$ to state $-$. We can take advantage from the fact that for a conjunction of equations, the letters for which there is a transition will not coincide in general. Instead of computing the automaton for each single equation in a conjunction and then compute the product of the automata, we will directly compute the automaton for the conjunction.

Consider the system of equations $S$:

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1n}x_n & = & k_1 \\ a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2n}x_n & = & k_2 \\ \vdots & & & & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + \cdots + & a_{mn}x_n & = & k_m \end{array}$$

Let us compute the automaton accepting the solutions of $S$. The alphabet will be $\{0,1\}^n$. The states of the automaton (other than $[-]$) will be labeled by conjunctions of equations of the form

$$\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = k_i$$

When letter $b_1 \cdots b_n$ is read in such a state the remainder of the word must encode a solution of

$$\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij}(2x_j + b_j) = k_i$$

or, equivalently, of

$$\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} 2(a_{ij} x_j) = k_i - \Sigma_{j=1}^{n} a_{ij} b_j$$

This formula is obviously unsatisfiable if, for some $i$, $k_i$ and $\Sigma_{j=1}^{n} a_{ij} b_j$ do not have the same parity. On the other hand, if for $1 \leq i \leq m$, $k_i$ and $\Sigma_{j=1}^{n} a_{ij} b_j$ have the same parity, the formula is equivalent to

$$\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = \frac{k_i - \Sigma_{j=1}^{n} a_{ij} b_j}{2}$$

which is the new state reached in the automaton. This state has been obtained from the previous by adding a vector of integers to the vector of the constants in the right-hand sides before dividing them by 2. Let us call *increment* of letter $b_1 \cdots b_n$ the $m$-uple of integers

$$Incr(\alpha) = (\Sigma_{j=1}^{n} a_{1j} b_j, \ldots, \Sigma_{j=1}^{n} a_{mj} b_j)$$

If we define the *parity vector* of a state

$$\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = k_i$$

as the $i$-uple of bits $(k_1(\text{mod}2), \ldots, k_m(\text{mod}2))$, then we can compute in advance, for each of the $2^m$ possible parity vectors, the letters of $\{0,1\}^n$ for which there will be a transition to a state other than $[-]$, as well as the corresponding increment. This will ease the computation of the closure of the initial state for the states reached using the transition schemata:

**Transition**

$$[\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = k_i] \xrightarrow{\alpha} [\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = \frac{k_i - \Sigma_{j=1}^{n} a_{ij} b_j}{2}]$$

if $\alpha = b_1 b_2 \cdots b_n$ and for $1 \leq i \leq m$, $k_i - \Sigma_{j=1}^{n} a_{ij} b_j$ is even.

**Fail**

$$[\bigwedge_{1 \leq i \leq m} \Sigma_{j=1}^{n} a_{ij} x_j = k_i] \xrightarrow{\alpha} [-]$$

if $\alpha = b_1 b_2 \cdots b_n$ and $k_i - \Sigma_{j=1}^{n} a_{ij} b_j$ is odd for some $i$.

*Example 1.* Consider the following system $S$ of equations (the problem of the ape and the coconuts):

$$\begin{array}{llllll} -4x_1 & +5x_2 & & & & = 1 \\ & -4x_2 & +5x_3 & & & = 1 \\ & & -4x_3 & +5x_4 & & = 1 \\ & & & -4x_4 & +5x_5 & = 1 \\ & & & & -4x_5 \; +5x_6 & = 1 \end{array}$$

Let us show how the transition scheme **Transition** can be efficiently implemented. The states of the automaton (other than $[-]$) will be labeled by conjunctions of equations of the form

$$\bigwedge_{1 \leq i \leq 5} -4x_i + 5x_{i+1} = k_i$$

The automaton for each of these equations will have $2^6$ transitions (by every letter of $\{0,1\}^6$) from each state. Half of them will lead to a state different than

| Parity of the state | Letter $\alpha$ | $Incr(\alpha)$ | Parity of the state | Letter $\alpha$ | $Incr(\alpha)$ |
|---|---|---|---|---|---|
| 0 0 0 0 0 | 1 0 0 0 0 0 | ( -4, 0, 0, 0, 0) | 1 0 0 0 0 | 1 1 0 0 0 0 | (1, -4, 0, 0, 0) |
| | 0 0 0 0 0 0 | ( 0, 0, 0, 0, 0) | | 0 1 0 0 0 0 | (5, -4, 0, 0, 0) |
| 0 0 0 0 1 | 1 0 0 0 0 1 | ( -4, 0, 0, 0, 5) | 1 0 0 0 1 | 1 1 0 0 0 1 | ( 1, -4, 0, 0, 5) |
| | 0 0 0 0 0 1 | ( 0, 0, 0, 0, 5) | | 0 1 0 0 0 1 | ( 5, -4, 0, 0, 5) |
| 0 0 0 1 0 | 1 0 0 0 1 0 | ( -4, 0, 0, 5, -4) | 1 0 0 1 0 | 1 1 0 0 1 0 | ( 1, -4, 0, 5, -4) |
| | 0 0 0 0 1 0 | ( 0, 0, 0, 5, -4) | | 0 1 0 0 1 0 | ( 5, -4, 0, 5, -4) |
| 0 0 0 1 1 | 1 0 0 0 1 1 | ( -4, 0, 0, 5, 1) | 1 0 0 1 1 | 1 1 0 0 1 1 | ( 1, -4, 0, 5, 1) |
| | 0 0 0 0 1 1 | ( 0, 0, 0, 5, 1) | | 0 1 0 0 1 1 | ( 5, -4, 0, 5, 1) |
| 0 0 1 0 0 | 1 0 0 1 0 0 | ( -4, 0, 5, -4, 0) | 1 0 1 0 0 | 1 1 0 1 0 0 | ( 1, -4, 5, -4, 0) |
| | 0 0 0 1 0 0 | ( 0, 0, 5, -4, 0) | | 0 1 0 1 0 0 | ( 5, -4, 5, -4, 0) |
| 0 0 1 0 1 | 1 0 0 1 0 1 | ( -4, 0, 5, -4, 5) | 1 0 1 0 1 | 1 1 0 1 0 1 | ( 1, -4, 5, -4, 5) |
| | 0 0 0 1 0 1 | ( 0, 0, 5, -4, 5) | | 0 1 0 1 0 1 | ( 5, -4, 5, -4, 5) |
| 0 0 1 1 0 | 1 0 0 1 1 0 | ( -4, 0, 5, 1, -4) | 1 0 1 1 0 | 1 1 0 1 1 0 | ( 1, -4, 5, 1, -4) |
| | 0 0 0 1 1 0 | ( 0, 0, 5, 1, -4) | | 0 1 0 1 1 0 | ( 5, -4, 5, 1, -4) |
| 0 0 1 1 1 | 1 0 0 1 1 1 | ( -4, 0, 5, 1, 1) | 1 0 1 1 1 | 1 1 0 1 1 1 | ( 1, -4, 5, 1, 1) |
| | 0 0 0 1 1 1 | ( 0, 0, 5, 1, 1) | | 0 1 0 1 1 1 | ( 5, -4, 5, 1, 1) |
| 0 1 0 0 0 | 1 0 1 0 0 0 | ( -4, 5, -4, 0, 0) | 1 1 0 0 0 | 1 1 1 0 0 0 | ( 1, 1, -4, 0, 0) |
| | 0 0 1 0 0 0 | ( 0, 5, -4, 0, 0) | | 0 1 1 0 0 0 | ( 5, 1, -4, 0, 0) |
| 0 1 0 0 1 | 1 0 1 0 0 1 | ( -4, 5, -4, 0, 5) | 1 1 0 0 1 | 1 1 1 0 0 1 | ( 1, 1, -4, 0, 5) |
| | 0 0 1 0 0 1 | ( 0, 5, -4, 0, 5) | | 0 1 1 0 0 1 | ( 5, 1, -4, 0, 5) |
| 0 1 0 1 0 | 1 0 1 0 1 0 | ( -4, 5, -4, 5, -4) | 1 1 0 1 0 | 1 1 1 0 1 0 | ( 1, 1, -4, 5, -4) |
| | 0 0 1 0 1 0 | ( 0, 5, -4, 5, -4) | | 0 1 1 0 1 0 | ( 5, 1, -4, 5, -4) |
| 0 1 0 1 1 | 1 0 1 0 1 1 | ( -4, 5, -4, 5, 1) | 1 1 0 1 1 | 1 1 1 0 1 1 | ( 1, 1, -4, 5, 1) |
| | 0 0 1 0 1 1 | ( 0, 5, -4, 5, 1) | | 0 1 1 0 1 1 | ( 5, 1, -4, 5, 1) |
| 0 1 1 0 0 | 1 0 1 1 0 0 | ( -4, 5, 1, -4, 0) | 1 1 1 0 0 | 1 1 1 1 0 0 | ( 1, 1, 1, -4, 0) |
| | 0 0 1 1 0 0 | ( 0, 5, 1, -4, 0) | | 0 1 1 1 0 0 | ( 5, 1, 1, -4, 0) |
| 0 1 1 0 1 | 1 0 1 1 0 1 | ( -4, 5, 1, -4, 5) | 1 1 1 0 1 | 1 1 1 1 0 1 | ( 1, 1, 1, -4, 5) |
| | 0 0 1 1 0 1 | ( 0, 5, 1, -4, 5) | | 0 1 1 1 0 1 | ( 5, 1, 1, -4, 5) |
| 0 1 1 1 0 | 1 0 1 1 1 0 | ( -4, 5, 1, 1, -4) | 1 1 1 1 0 | 1 1 1 1 1 0 | ( 1, 1, 1, 1, -4) |
| | 0 0 1 1 1 0 | ( 0, 5, 1, 1, -4) | | 0 1 1 1 1 0 | ( 5, 1, 1, 1, -4) |
| 0 1 1 1 1 | 1 0 1 1 1 1 | ( -4, 5, 1, 1, 1) | 1 1 1 1 1 | 1 1 1 1 1 1 | ( 1, 1, 1, 1, 1) |
| | 0 0 1 1 1 1 | ( 0, 5, 1, 1, 1) | | 0 1 1 1 1 1 | ( 5, 1, 1, 1, 1) |

**Fig. 2.** The “schematic” transition table for the problem $\bigwedge_{1\leq i\leq 5} -4x_i + 5x_{i+1} = 1$. The initial state has parity vector 11111. Let us denote the state $[\bigwedge_{1\leq i\leq 5} -4x_i + 5x_{i+1} = k_i]$ by $(k_1, \ldots, k_5)$. The state accessed from $(k_1, \ldots, k_5)$ by by a letter $\alpha$ (corresponding to the parity vector of $(k_1, \ldots, k_5)$ in the table) is $\frac{(k_1,\ldots,k_5)+incr(\alpha)}{2}$. For instance, in the initial state, $(1, 1, 1, 1, 1)$ and there is a transition by 111111 to itself and by 011111 to $(3, 1, 1, 1, 1)$. The closure of the initial state by the **Transition** scheme contains 9524 states (including (0,0,0,0,0) : the problem is satisfiable). It is computed in LISP in less than half a minute.

[$-$]. Figure 2 gives a compact representation of the **Transition** scheme for this problem. Only two transitions lead to a state other than [$-$] for each possible parity vector.

### 2.3 Complexity

**Proposition 3.** *Let $\phi$ be the equation $a_1x_1+\cdots+a_nx_n = k$. Let $K(\phi)$ be the sum of the sizes of $|a_1|, \ldots, |a_n|, |k|$, written in binary, $V(\phi)$ the number $n$ of variables of $\phi$ and $|\phi| = K(\phi) + V(\phi)$. A minimal, deterministic, complete automaton $A$ recognizing the solutions of $\phi$ can be constructed in time $O(2^{|\phi|})$. The number of states of $A$ is bounded by $2^{K(\phi)}$ and the number of transitions from each state is $2^{V(\phi)}$.*

*Proof.* First, the number of transitions from any state in a complete automaton is the cardinal of the alphabet, in our case $2^{V(\phi)}$. We have seen in section 1 that by reading letter $\alpha = b_1 \cdots b_{V(\phi)}$ in a state $[a_1x_1 + \cdots + a_nx_n = k]$, the new state is labeled by $-$, or by the equation

$$a_1x_1 + \cdots + a_nx_n = \frac{k - \Sigma_{i=1}^{V(\phi)} b_i a_i}{2}$$

As soon as $|k| > |\Sigma_{i=1}^{V(\phi)} b_i a_i|$, then $|k| < |\frac{k-\Sigma_{i=1}^{V(\phi)} b_i a_i}{2}|$. Hence the number of states different than [$-$] is bounded by $|k| + \Sigma_{i=1}^{V(\phi)} |a_i|$ which is exactly $2^{K(\phi)}$. The method that we have given computes directly a deterministic complete automaton. We have seen that it can always be minimalized in linear time, hence the result.□

The same result holds for disequations (only the accepting and non-accepting states are permuted), and for inequations (the same bound is valid, the constant on the right-hand side of the label of the new state is the integer part of $\frac{k-\Sigma_{i=1}^{V(\phi)} b_i a_i}{2}$). Deciding arbitrary unquantified formulae can be done with the same complexity, but disjunctions introduce some non-determinism.

**Proposition 4.** *Let $\phi$ be an unquantified formula of Presburger arithmetic built up using equations, disequations, inequations and the connectives $\vee$ and $\wedge$. Let $C(\phi)$ be the number of connectives in $\phi$, and $|\phi| = C(\phi) + |\phi_1| + \cdots + |\phi_n|$, where $\phi_1, \ldots, \phi_n$ are the atomic subformulae of $\phi$ and $|\phi_i|$ is defined as in the previous proposition.*

*A complete, non-deterministic automaton $A$ recognizing the solutions of $\phi$ can be constructed in time $O(2^{|\phi|})$. The size $|A|$ of $A$ is the number of states plus the number of transitions. The emptyness of the language recognized by $A$ can be decided in time $O(|A|)$.*

*Proof.* By induction on the number of connectives. □

## 3 Presburger arithmetic

As already recalled, the decidability of Presburger's arithmetic follows from Büchi's theorem. However, in principle, the derived decision procedure might require non-ELEMENTARY time; as for Büchi's theorem, eliminating existential quantifiers can be done in linear time, simply by removing the corresponding component in the transition rules. However, each universal quantifier may actually require (at least) a determinization step: each universal quantifier elimination might yield an exponential blowup of the automaton. (As in S1S, or for the inequivalence problem of regular expressions see [6, ?]). However, in our context, the automata are not just any automata: there is a superexponential bound on the number of states of a minimal deterministic automaton accepting a formula:

**Lemma 5.** *Let $\phi \equiv Q_n x_n, \ldots, Q_1 x_1 \;\; \psi$ be a formula of Presburger arithmetic where $Q_i$ is either $\exists$ or $\forall$ and $\psi$ is an unquantified formula with variables $x_1, \ldots, x_n, y_1, \ldots, y_m$. There is a deterministic and complete automaton recognizing the solutions of $\phi$ with at most $O(2^{2^{2^{|\phi|}}})$ states.*

*Proof.* As before, the states will be (labeled by) formulae. These formulae will always be of the form $Q_n x_n, \ldots, Q_1 x_1 \;\; B(\psi_1, \ldots, \psi_k)$ where, for each $i$, $\psi_i$ is a state of the automaton accepting the solutions of $\psi$ (as computed in previous section) and $B(\cdot, \ldots, \cdot)$ is a Boolean combination involving the connectives $\vee$ and $\wedge$.

The initial state is $[\phi]$.

If the quantifiers range over naturals, the two following rules are trivially correct:

∀-bit
$$\forall x \; \phi \quad \rightarrow \quad \forall x \;\; (\phi\{x \mapsto 2x\} \;\wedge\; \phi\{x \mapsto 2x+1\})$$

∃-bit
$$\exists x \; \phi \quad \rightarrow \quad \exists x \;\; (\phi\{x \mapsto 2x\}_1^0 \phi\{x \mapsto 2x+1\})$$

If $\alpha \in \{0,1\}^m$ is read in a state $[Q_n x_n, \ldots, Q_1 x_1 \;\; B(\psi_1, \ldots, \psi_k)]$, then the remainder of the word must satisfy the formula

$$Q_n x_n, \ldots, Q_1 x_1 \;\; B(\psi_1\{y_1 \mapsto 2y_1 + b_1, \ldots, y_m \mapsto 2y_m + b_m\}, \ldots, \psi_k\{y_1 \mapsto 2y_1 + b_1, \ldots, y_m \mapsto 2y_m + b_m\})$$

where $\alpha = b_1 \cdots b_m$.

If we apply the rule **∀-bit** or **∃-bit** to $Q_1, \ldots, Q_n$ successively in $Q_n x_n, \ldots, Q_1 x_1 B(\psi_1, \ldots, \psi_n)$, we obtain a formula

$$Q_n x_n, \ldots, Q_1 x_1 \;\; B'(\cdots, \psi_i \otimes \beta_j, \cdots)$$

in which the quantifier prefix has not changed but we have another Boolean combination over atomic formulas of the form $\psi_i \otimes \beta_j$ ($\beta_j \in \{0,1\}^{n+m}$), which is again a state of the automaton accepting $\psi$.

The maximal number of such states is the number of propositional formulae over the alphabet $\psi \otimes \beta_1 \ldots \otimes \beta_j$, i.e. the states of the automaton accepting the solutions of $\psi$. By proposition 2.3, there are at most $2^{|\psi|}$ such states, hence at most $2^{2^{2^{|\psi|}}}$ propositional formulae over these states. This gives the desired bound.

The final states of the automaton are the formulae $\Psi$ such that $0 \cdots 0 \models \Psi$. Note however, that we need not effectively to compute these final states; our proof is not constructive yet. □

This existence lemma gives an upper bound for all intermediate automata which are computed using the standard Büchi's technique applied to our problem:

**Theorem 6.** *The automaton algorithm of section 1.2 decides Presburger arithmetic in triple exponential time.*

*Proof.* Using the algorithm of section 1.2 and minimizing the automaton after each quantifier elimination, each intermediate automaton is smaller than any deterministic automaton accepting the same language, hence, by previous lemma has at most a triple exponential size. The time required for the computation is a polynomial in this maximal size. □

## 4 Conclusion

## References


1. Evelyne Contejean and Hervé Devie. An efficient algorithm for solving systems of diophantine equations. *Information and Computation*, 113(1):143–172, August 1994.
2. Elliot. On linear homogeneous diophantine equations. *Quartely J. of Pure and Apllied Maths*, 136, 1903.
3. A. Fortenbacher. An algebraic approach to unification under associativity and commutativity. In *Proc. Rewriting Techniques and Applications 85, Dijon, LNCS 202*. Springer-Verlag, May 1985.
4. P. MacMahon. *Combinatory Analysis*, volume 2, chapter II: A Syzygetic Theory, pages 111–114. Reprinted by Chelsea in 1960, 1919.
5. M. Presburger. Über die Vollständingen einer gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du premier Congrès des Mathématiciens des Pays slaves*, Warszawa, 1929.
6. L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 1–9, 1973.
7. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 134–191. Elsevier, 1990.


This article was processed using the LaTeX macro package with LLNCS style