Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Complexity of equations over sets of natural numbers

Artur Jeż · Alexander Okhotin

the date of receipt and acceptance should be inserted later

Abstract Systems of equations of the form $X_i = \varphi_i(X_1, \ldots, X_n)$ $(1 \leqslant i \leqslant n)$ are considered, in which the unknowns are sets of natural numbers. Expressions φ_i may contain the operations of union, intersection and elementwise addition $S+T = \{m+n \mid m \in S, n \in T\}$. A system with an EXPTIME-complete least solution is constructed in the paper through a complete arithmetization of EXPTIME-completeness. At the same time, it is established that least solutions of all such systems are in EXPTIME. The general membership problem for these equations is proved to be EXPTIME-complete. Among the consequences of the result is EXPTIME-completeness of the compressed membership problem for conjunctive grammars.

1 Introduction

The study of expressions over sets of numbers and of the computational complexity of their properties began in the seminal paper by Stockmeyer and Meyer [24], who considered formal languages over a one-letter alphabet as subsets of $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$, thus turning a concatenation of languages into an elementwise addition of sets: $S + T = \{m + n \mid m \in S, n \in T\}$. Stockmeyer and Meyer established, in particular, that the membership problem for expressions with union and addition is NP-complete, and if the expressions may also contain complementation, then the problem becomes PSPACE-complete. Some follow-up work was done by Yang [26], who considered *circuits* computing sets of numbers (that is, expressions in which subexpressions may be shared) with an extra operation of elementwise multiplication, and established similar

A preliminary version of this paper was presented at STACS 2008 conference held in Bordeaux, France on 21–23 February, 2008.

A. Jeż
Institute of Computer Science, University of Wrocław, Poland, E-mail: aje@ii.uni.wroc.pl
Supported by MNiSW grant N N206 259035 2008–2010.

A. Okhotin
Academy of Finland
Department of Mathematics, University of Turku, Finland, E-mail: alexander.okhotin@utu.fi
Supported by the Academy of Finland under grant 118540.

complexity results. A systematic study of complexity of expressions and circuits with different sets of operations was recently carried out by McKenzie and Wagner [15].

This work has inspired some related studies. Breunig [2] investigated the same formalisms defined over sets of positive integers and showed that in some cases the membership problem becomes computationally easier. Similarly, the complexity of the membership problem for expressions and circuits over sets of any integers was studied by Travers [25], who found cases where the problem becomes harder and also cases where it becomes easier. Glaßer et al. [6] determined the complexity of the equivalence problem for the classes studied by McKenzie and Wagner [15].

This paper considers a different extension of this model: the *equations* over sets of natural numbers. These are systems of equations of the resolved form

$$\begin{cases} X_1 = \varphi_1(X_1, \ldots, X_n) \\ \quad \vdots \\ X_n = \varphi_n(X_1, \ldots, X_n) \end{cases} \tag{*}$$

in which every variable X_i represents an unknown set of nonnegative integers. The right-hand side φ_i of each equation may contain the operations of union, intersection and addition, as well as singleton constants. Every such system has a least solution with respect to componentwise inclusion, which can be obtained by fixpoint iteration. The X_1-component of this least solution is regarded as the set of numbers defined by the system, and thus the membership problem for such systems is considered similarly to expressions and circuits.

While the expressions defined by Stockmeyer and Meyer [24] represent regular expressions over a one-letter alphabet, equations over sets of numbers naturally correspond to *language equations* over the same alphabet $\{a\}$. Language equations have recently become an active area of research, with unexpected hardness results for them obtained by Kunc [11] and by Okhotin [20,21]; for more details the reader is referred to a recent survey by Kunc [12]. Systems of the particular form (*) using union and concatenation of languages are well-known to represent context-free grammars [5], while the variant that allows intersection similarly represents *conjunctive grammars*, a generalization of the context-free grammars introduced by Okhotin [17]. The expressive power of conjunctive grammars over a one-letter alphabet has been understood only recently, with the first grammar for a non-regular language constructed by Jeż [8], and with a large class of representable languages subsequently obtained by Jeż and Okhotin [9]. These results, which immediately apply to systems of equations (*) over sets of numbers, provide a technical basis for the present work.

The core result of this paper, established in Section 3, is a construction of a fixed system of the form (*), such that testing the membership of numbers in its least solution is an EXPTIME-hard problem, with the numbers given in binary notation. The result is obtained by a new kind of arithmetization of an alternating linear-space Turing machine, with numbers representing its instantaneous descriptions. It is also shown that for every system (*), the membership of numbers in its least solution can be tested in exponential time, which makes the constructed set the hardest. This yields EXPTIME-completeness of the *fixed membership problem*, in which the system is not a part of the input. Comparing this with circuits over sets of numbers, the latter may only generate ultimately periodic sets (unless multiplication of sets is employed), and thus the computational complexity of those sets is trivial.

The cited results on expressions and circuits over sets of numbers [2, 15, 24, 25, 26] were concerned with the complexity of the *general membership problem*, where both a circuit and a number are given as an input. In those cases it was sufficient to encode an instance of some hard problem in a circuit, which in most cases allows natural reductions from existing problems. The task approached in this paper is more difficult: a fixed system of equations representing a problem has to be constructed, and then another problem has to be reduced to it by encoding its instances as numbers. This requires a complete arithmetization of EXPTIME-completeness.

This result easily leads to the complexity of the general membership problem for equations with union, intersection and addition, which is stated as follows: *"Given a system (*) and a number $n \geqslant 0$ in binary notation, determine whether n is in the first component of the least solution of the system"*. For integer expressions and integer circuits with the same operations on sets, which can be regarded as an acyclic case of systems (*), it is known from McKenzie and Wagner [15] that the membership problem is PSPACE-complete. Another weaker model are systems (*) with union and addition, that is, without intersection, for which the corresponding problem is NP-complete due to the result of Huynh [7] on the commutative case of the context-free grammars. For equations with union, intersection and addition, the general membership problem is shown to be EXPTIME-complete in Section 4.

The results of this paper have a few implications on conjunctive grammars worked out in Section 5. An immediate consequence is that there exists a conjunctive grammar over an alphabet $\{a\}$ generating *unary* representations of all numbers from an EXPTIME-complete set of their *binary* representations. This unary conjunctive language may be regarded as the computationally hardest of its kind; note that it is unlikely that conjunctive grammars define any P-complete languages, as, under the standard complexity-theoretic assumptions, no sparse P-complete languages exist [16, 3], and unary languages are sparse.

Another implication refers to the complexity of the *compressed membership problem* for conjunctive grammars, in which the input string w is supplied as a context-free grammar G_w generating the singleton language $\{w\}$. For regular expressions and for deterministic finite automata, this problem is P-complete by the results of Plandowski and Rytter [22] and Markey and Schnoebelen [14]. For context-free grammars, it is PSPACE-complete due to Plandowski and Rytter [22] and Lohrey [13]. Lohrey [13] has also established the EXPSPACE-completeness of the problem for context-sensitive grammars. In this paper, conjunctive languages are put in the context of the cited research by showing that their compressed membership problem is EXPTIME-complete.

2 Language equations and conjunctive grammars

In language equations, the unknowns are formal languages over an alphabet Σ. If $|\Sigma| = 1$, they coincide with equations over sets of numbers, while for larger alphabets they constitute a more general notion. The main object of this study are equations of the resolved form (*), in which variables assume values of sets of non-negative integers, and the right-hand sides may contain the operations of union, intersection and addition of sets, as well as singleton constant sets. These equations obviously correspond to *language equations* over a one-letter alphabet with the operations of union, intersection and concatenation, and the recent results on language equations

of this kind provide a theoretical foundation, as well as another motivation, for the present research.

The first type of language equations to be studied were systems of the same form $X_i = \varphi_i(X_1, \ldots, X_n)$ ($i \in \{1, \ldots, n\}$) containing union and concatenation, but no intersection: Ginsburg and Rice [5] established that these equations provide a natural semantics for the context-free grammars. To be precise, every such system has solutions, and among them there is always a *least solution*, which is a componentwise intersection of all solutions. Furthermore, the least solution can be obtained by fixpoint iteration as

$$\bigsqcup_{i \geqslant 0} \varphi^i(\varnothing, \ldots, \varnothing), \tag{1}$$

where φ is the right-hand side of the system as an operator on the set of n-tuples of languages, while $\sqcup$ denotes componentwise union of n-tuples of languages.

Equations augmented with an intersection operation inherit this property, and they can be used to define the following generalization of the context-free grammars.

Definition 1 (Okhotin [17]) A conjunctive grammar is a quadruple $G = (\Sigma, N, P, S)$, in which Σ and N are disjoint finite non-empty sets of terminal and nonterminal symbols respectively; P is a finite set of grammar rules, each of the form

$$A \to \alpha_1 \& \ldots \& \alpha_n \quad (\text{where } A \in N,\ n \geqslant 1 \text{ and } \alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*)$$

while $S \in N$ is a nonterminal designated as the start symbol.

The semantics of a conjunctive grammar is defined by the least solution of the following system of language equations, in which nonterminal symbols are variables:

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in P} \bigcap_{i=1}^{m} \alpha_i \quad (\text{for all } A \in N) \tag{2}$$

Each string $\alpha_i \in (\Sigma \cup N)^*$ represents a concatenation of variables and singleton constants. The component of the least solution corresponding to each $A \in N$ is then denoted by $L_G(A)$, and $L(G)$ is defined as $L_G(S)$.

An equivalent definition of conjunctive grammars can be given using term rewriting [17], which generalizes Chomsky's string rewriting. The importance of these grammars lies with the fact that their expressive power is substantially greater than that of the context-free grammars, while the generated languages can still be parsed in time $O(n^3)$. Furthermore, the practical context-free parsing algorithms, such as the generalized LR and the recursive descent, admit generalization to conjunctive grammars without an increase in their computational complexity.

For a one-letter alphabet $\Sigma = \{a\}$, a system of language equations (2) can be regarded as a system of equations (*) over sets of natural numbers using union, intersection and addition. The question of whether conjunctive grammars can generate any non-regular unary languages has been an open problem for some years, until recently solved by Jeż [8], who constructed a grammar for the language $\{a^{4^n} \mid n \geqslant 0\}$. This grammar is given below, along with its reformulation as a resolved system of four equations over sets of numbers:

Example 1 (Jeż [8]) The following conjunctive grammar with the start symbol A_1 generates the language $\{a^{4^n} \mid n \geqslant 0\}$.

$$\begin{array}{ll} A_1 \to A_2A_2 \& A_1A_3 \mid a & X_1 = \big((X_2 + X_2) \cap (X_1 + X_3)\big) \cup \{\mathtt{1}\} \\ A_2 \to A_{12}A_2 \& A_1A_1 \mid aa & X_2 = \big((X_{12} + X_2) \cap (X_1 + X_1)\big) \cup \{\mathtt{2}\} \\ A_3 \to A_{12}A_{12} \& A_1A_2 \mid aaa & X_3 = \big((X_{12} + X_{12}) \cap (X_1 + X_2)\big) \cup \{\mathtt{3}\} \\ A_{12} \to A_3A_3 \& A_1A_2 & X_{12} = (X_3 + X_3) \cap (X_1 + X_2) \end{array}$$

The corresponding system of equations over sets of numbers has the least solution $X_i = \{\ell \mid \text{base-4 notation of } \ell \text{ is } i\mathtt{0}\ldots\mathtt{0}\}$ for $i = 1, 2, 3, 12$, where X_1 is the set of all powers of 4.

Sets of this kind can be conveniently specified by regular expressions for the corresponding sets of base-k notations of numbers, which in this case are $\mathtt{10}^*$, $\mathtt{20}^*$, $\mathtt{30}^*$ and $\mathtt{120}^*$, respectively.

Using the same technique as in the above example in a more elaborate construction, a general theorem on the expressive power of unary conjunctive grammars was established. It can be reformulated for equations over sets of numbers as follows:

Theorem 1 (Jeż [8]) *For every $k \geqslant 2$ and for every finite automaton M over the alphabet $\{0, \ldots, k-1\}$ there exists a resolved system of language equations over sets of numbers using union, intersection and addition, with the least solution*

$$(S_1, S_2, \ldots, S_n),$$

where $S_i \subseteq \mathbb{N}_0$ and $S_1 = \{\ell \mid k\text{-ary notation of } \ell \text{ is in } L(M)\}$.

A generalization of this result to a larger family of automata recognizing positional notations was established in a recent paper by Jeż and Okhotin [9].

In terms of language theory, representing all sets of numbers with a regular positional notation already constitutes a substantial expressive power. However, it has no implications on the computational complexity, as all these sets are computationally easy. The more general representation theorem of Jeż and Okhotin [9] also does not imply any better complexity results than P-completeness, which, as the present paper shows, is much below the actual complexity of these equations. Therefore, a new method of constructing such equations is needed to understand their complexity. This step is made in the next section, which introduces an arithmetization technique based upon addition of sets of numbers.

3 Arithmetization of EXPTIME-completeness

The core result of this paper is a construction of a particular resolved system of equations over sets of numbers, for which testing membership of numbers in its least solution is an EXPTIME-complete problem. It is accompanied by a matching upper bound:

Theorem 2 *The family of sets of numbers representable by resolved systems of equations with union, intersection and addition, as well as singleton constants, is a subset of EXPTIME and contains an EXPTIME-complete language.*

The proof is by constructing a system of equations that encodes a computation of any linearly bounded alternating Turing machine (ATM). It is known that such machines recognize some EXPTIME-complete sets [4].

Furthermore, an ATM shall operate on a *circular tape* and move to the right at every step. Its tape shall originally contain the input string, and the cells containing it constitute all space available to the machine. Such a machine can simulate an arbitrary linear-bounded ATM by marking the position of its head on the tape, and by making one transition of the simulated ATM per each traversal of the circular tape. Hence, these restricted ATMs are as powerful as linear-bounded ATMs of the general form.

Formally, such a machine is defined as $M = (\Omega, \Gamma, Q_E, Q_A, \delta, q_0, q_{fin})$, where Ω is the input alphabet, $\Gamma = \{a_0, a_1, \ldots, a_{\max}\} \supset \Omega$ is the tape alphabet, Q_E and Q_A are disjoint sets of existential and universal states, respectively, $Q = Q_E \cup Q_A = \{q_0, q_1, \ldots, q_{max}\}$ and $q_{fin} \in Q$. Given an input $w \in \Omega^+$, M starts in state q_0 with the head over the first symbol of w. The transition function is $\delta : Q \times \Gamma \to 2^{Q \times \Gamma}$, and the head is moved one symbol to the right at every step. Once the head moves beyond the right-most symbol, it is moved back to the first symbol of w, maintaining its current state; this implements a circular tape. For technical reasons, assume that $(q, a') \notin \delta(q, a)$ for all $q \in Q$ and $a, a' \in \Gamma$ (that is, the machine never stays in the same state), and that $\delta(q, a) \neq \varnothing$ for all $q \in Q_A$ and $a \in \Gamma$.

The construction of a system of equations over sets of numbers simulating a computation is based upon representing instantaneous descriptions of the ATM as *numbers*. These numbers shall be considered in positional notation with base $8+|Q|+\max(|Q|+7, |\Gamma|)$, and the entire argument is based upon mapping the symbols used by the machine to digits, and then using addition to manipulate individual digits in the positional notation of numbers. It should be noted that this positional notation is only a tool for understanding the constructions, while the actual equations, by definition, deal with numbers as they are.

Let $\Sigma = \{\mathtt{0}, \mathtt{1}, \ldots, \mathtt{7}+|Q|+\max(|Q|+7, |\Gamma|)\}$ be the alphabet of digits, and define the mapping of symbols to digits, $\langle\cdot\rangle : Q \cup \Gamma \to \Sigma$, as follows:

$$\langle q_i \rangle = \mathtt{7} + i \quad (\text{for } q_i \in Q),$$
$$\langle a_i \rangle = \mathtt{7} + |Q| + i \quad (\text{for } a_i \in \Gamma).$$

The notation $\langle\cdot\rangle$ is naturally extended to strings over $Q \cup \Gamma$ by $\langle s_1 \ldots s_\ell \rangle = \langle s_1 \rangle \ldots \langle s_\ell \rangle$. Furthermore, let $\langle Q \rangle = \{\langle q \rangle \,|\, q \in Q\}$ and $\langle \Gamma \rangle = \{\langle a \rangle \,|\, a \in \Gamma\}$. Now the *tape* of the ATM containing symbols $a_{i_1} \ldots a_{i_n}$, with the head over the j-th symbol and the machine in state q, is represented as the following string of digits:

$$\mathtt{0}\langle a_{i_1} \rangle \ldots \mathtt{0}\langle a_{i_{j-1}} \rangle \langle q \rangle \langle a_{i_j} \rangle \mathtt{0} \langle a_{i_{j+1}} \rangle \ldots \mathtt{0}\langle a_{i_n} \rangle \mathtt{0} \in \Sigma^*$$

For technical reasons, configurations, in which the head has just moved over the last symbol but has not yet jumped to the first position, are considered separately and will be represented as strings of the form

$$\mathtt{0}\langle a_{i_1} \rangle \ldots \mathtt{0}\langle a_{i_n} \rangle \langle q \rangle,$$

where q is the current state. Note that digits denoting letters are written only in even positions, while odd positions are reserved for the states of the Turing machine. The set of all strings of digits representing valid encodings of the tape is specified by the following regular expression over Σ:

$$\text{Tape} = (\mathtt{0}\langle \Gamma \rangle)^* \langle Q \rangle (\langle \Gamma \rangle \mathtt{0})^* \setminus \langle Q \rangle$$

The set Tape should be considered as a formal language over Σ, which will be used later as a part of representations of some sets of numbers. Subsets of this set representing tapes with different states will be denoted as follows:

$$\text{Tape}_\alpha = \{w \mid w \in \text{Tape},\ \alpha \in (\Gamma \cup Q)^*,\ \langle\alpha\rangle \text{ is a substring of } w\},$$
$$\text{Tape}^\ell_\alpha = \{w \mid w \in \text{Tape},\ \alpha \in (\Gamma \cup Q)^*,\ \langle\alpha\rangle \text{ is a prefix of } w\}.$$

Besides the contents of the tape, the encoding for Turing machine configurations uses a *counter of rotations* of the circular tape. This counter specifies the number of circles through the tape the machine is still allowed to make before it must halt. It is represented in binary notation using digits $\{\mathtt{0},\mathtt{1}\}$, and the set of valid counter representations is

$$\text{Counter} = \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*,$$

where the digits are still in base-$|\Sigma|$ notation. Normally, the counter uses only digits $\{\mathtt{0},\mathtt{1}\}$, but in order to implement the incrementation of the counter, strings with one digit $\mathtt{2}$ representing zero with carry shall be used as well. The set of valid representations of counters with a carry is

$$\text{Counter}' = \mathtt{1}\{\mathtt{0},\mathtt{1}\}^*\mathtt{20}^* \cup \mathtt{20}^*.$$

For every string $c_{k-1}\ldots c_0 \in \text{Counter} \cup \text{Counter}'$, define its value as

$$\text{Value}(c_{k-1}\ldots c_0) = \sum_{j=0}^{k-1} c_j \cdot 2^j.$$

Now define the mapping from configurations of the Turing machine to numbers. A configuration with the tape contents, the head position and the current state given by a string of digits $w \in \text{Tape}$ and with the counter value given by $x \in \text{Counter}$ is represented by a string of digits

$$x\mathtt{55}w,$$

where two marker digits $\mathtt{55}$ separate the counter from the tape. This string of digits in base-$|\Sigma|$ positional notation specifies a certain number, which accordingly represents the configuration.

The key property of this encoding is that *every transition of the machine reduces the numerical value of its configuration.* Indeed, if the head is moved to the right, then a digit $\langle q\rangle$ is replaced with $\mathtt{0}$ and all other modifications are done on less significant digits. If the head jumps from the end to the beginning, then the counter is decremented, and since the counter occupies higher positions in the notation of the number than the tape, this transition decreases the value of the configuration as well. Such a monotonicity allows encoding the dependence of configurations on each other by using addition of nonnegative numbers only. This dependence is inductively expressed in the equations defined below.

The construction of a system of equations representing the computation of the ATM begins with some expressions that will be used in the right-hand sides of equations. These expressions contain some constant sets of numbers given as regular languages over the alphabet Σ. Every such language represents the set of all numbers with $|\Sigma|$-ary notation of the given form. According to Theorem 1, every such set can be represented by a separate system of equations using only singleton constants. All these subsystems

are assumed to be included in the constructed system, and each of the regular expressions in the system can be formally regarded as a reference to one of the auxiliary variables.

Definitions of a few of these regular languages incorporate positional notations of numbers obtained by subtracting one number from another. For convenience, these values are given in the form $u \boxminus v$, with $u, v \in \Sigma^*$ being positional notations of two numbers (the former shall be greater or equal to the latter). One can write, e.g., $(u \boxminus v)\mathtt{0}^*$ for the set of all numbers with their $|\Sigma|$-ary notation beginning with the fixed digits determined by the given difference, followed with any number of zeroes.

Under these conventions, the following four expressions are defined, each representing a function of one set argument:

$$\begin{aligned}
\mathrm{Step}(X) = {} & \bigcup_{\substack{q \in Q_E \\ a \in \Gamma}} \bigcup_{(q',a') \in \delta(q,a)} \mathrm{Move}_{q',a',q,a}(X) \\
& \cup \bigcup_{\substack{q \in Q_A \\ a \in \Gamma}} \bigcap_{(q',a') \in \delta(q,a)} \mathrm{Move}_{q',a',q,a}(X) \\
\mathrm{Move}_{q',a',q,a}(X) = {} & \Big[\big(X \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{a'q'}\big) + \big(\langle q\rangle\langle a\rangle\mathtt{0} \boxminus \langle a'\rangle\langle q'\rangle\big)(\mathtt{00})^* \Big] \\
& \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{qa} \\
\mathrm{Jump}(X) = {} & \bigcup_q \Big[\big(X \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_q^{\ell}\big) + (\mathtt{1000} \boxminus \langle q\rangle)(\mathtt{00})^+ + \langle q\rangle \Big] \\
& \cap (\mathrm{Counter} \cup \mathrm{Counter}')\mathtt{55}\,\mathrm{Tape}_q \\
\mathrm{Carry}(Y) = {} & \Big[\Big(\big[\big(Y \cap \{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape}\big) + \mathtt{10}^* \big] \cap \{\mathtt{0},\mathtt{1}\}^*\mathtt{3}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape} \Big) \\
& + \big(\mathtt{10} \boxminus \mathtt{3}\big)\mathtt{0}^* \Big] \cap \big(\{\mathtt{0},\mathtt{1}\}^+ \cup \{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\big)\,\mathtt{55}\,\mathrm{Tape}
\end{aligned}$$

Whenever the functions $\mathrm{Move}_{q',a',q,a}$ and Jump are applied to a set of configurations X, they manipulate the symbols in each configuration in this set in order to reconstruct the configuration at the previous step (one with a larger numerical value). In particular, Jump moves the head back over the edge of the tape, incrementing the counter, while $\mathrm{Move}_{q',a',q,a}$ reverses a transition from (q,a) to (q',a') by moving the head by one position to the left, restoring the letter a and returning to the state q. The function Step transcribes the logic of a single step of the ATM, taking the transition table and the alternation into account, while the function Carry is used to implement incrementation of the counter.

The set of final configurations of the machine is defined as follows:

$$\mathrm{Final} = \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{q_{fin}}.$$

The system of equations uses two variables, X and Y. Either variable represents the set of proper configurations of the machine, starting from which the machine accepts. The difference between these variables is that X represents configurations belonging to the set $\mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}$, while Y represents configurations from $(\mathrm{Counter} \cup \mathrm{Counter}')\mathtt{55}\,\mathrm{Tape}$, in which the counter may contain one carry digit $\mathtt{2}$ that needs to be propagated to higher positions. The equations, using the above auxiliary functions, are as follows:

$$\begin{cases} X = \mathrm{Final} \cup \mathrm{Step}(X) \cup \big(Y \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}\big) \\ Y = \mathrm{Jump}(X) \cup \mathrm{Carry}(Y) \end{cases} \tag{3}$$

Intuitively, the equation for X states that a configuration leads to acceptance if and only if it is itself accepting (Final), or one can directly proceed from it to a configuration leading to acceptance (Step(X)), or that it is a configuration obtained in Y. The equation for Y specifies circular rotation of the tape by Jump(X) and implements iterated carry propagation by a self-reference Carry(Y).

In order to determine the least solution of this system, let us first establish some properties of the auxiliary functions. The first quite elementary property is their distributivity over infinite union, which allows studying these operations as operations on individual numbers, and then infer their action on sets of numbers.

Lemma 1 (Distributivity) *Each function $f \in \{\mathrm{Move}_{q',a',q,a}, \mathrm{Jump}, \mathrm{Carry}\}$ is distributive over infinite union, in the sense that $f(S) = \bigcup_{n \in S} f(\{n\})$, for every $S \subseteq \mathbb{N}_0$.*

In the following, singleton arguments $\{n\}$ for the functions $\mathrm{Move}_{q',a',q,a}$, Jump and Carry shall be denoted without braces, as in Jump(n).

Lemma 1 follows from the fact that each of these expressions consists of intersections with constant sets, sums with constant sets and unions, for which this property holds in general:

Proposition 1 (Jeż, Okhotin [9]) *Let $\varphi(X)$ be an expression defined as a composition of the following operations: (i) the variable X; (ii) constant sets; (iii) union; (iv) intersection with a constant set; (v) addition of a constant set. Then φ is distributive over infinite union, that is, $\varphi(X) = \bigcup_{n \in X} \varphi(\{n\})$.*

On the other hand, note that if an expression contains an intersection or a sum of two expressions involving X, then it is not necessarily distributive over infinite union. In particular, Step need not be distributive.

A common expression used in these functions is addition of a constant set of numbers with $|\Sigma|$-ary notation $u\mathtt{0}^*$ (that is, a set $\left\{ m \cdot |\Sigma|^i \;\middle|\; i \geqslant 0 \right\}$) with one, two or three non-zero leading digits in u. The following lemma establishes that this addition can never rewrite the double markers $\mathtt{55}$, that is, every sum in which these markers are altered does not represent a valid tape contents. This means that such additions manipulate the counter and the tape separately, and the changes do not mix.

Lemma 2 (Marker preservation) *For every $x, x' \in \{\mathtt{0},\mathtt{1},\mathtt{2},\mathtt{3}\}^* \setminus \mathtt{0}\Sigma^*$ and $w, w' \in$ Tape, if $x'\mathtt{55}w' \in x\mathtt{55}w + (\Sigma^3 \cup \Sigma^2 \cup \Sigma)\mathtt{0}^*$, then $|w| = |w'|$.*

Proof Let $y = ijk\mathtt{0}^\ell$, with $i, j, k \in \Sigma$, be a string representing a number, and assume that $x'\mathtt{55}w' = x\mathtt{55}w + y$. The ℓ least significant digits of $x\mathtt{55}w$ and of $x'\mathtt{55}w'$ are then the same.

Consider the $(\ell+4)$-th digit of $x\mathtt{55}w$, let it be c. Since y has fewer than $\ell+4$ digits, any change at this position can only be due to a carry from the position $\ell+3$. As the digit $|\Sigma|-1$ is not used in any proper encoding, $c < |\Sigma| - 1$. Because the carry digit is at most 1, the $(\ell+4)$-th digit in $x'\mathtt{55}w'$ is less or equal to $c+1$, that is, it is less or equal to $|\Sigma|-1$. Therefore, there is no carry to the position $\ell+5$ in $x\mathtt{55}w+y$, and all digits in positions higher than $\ell+4$ in $x\mathtt{55}w+y$ are the same as in $x\mathtt{55}w$. Hence, $x'\mathtt{55}w'$ has at most four digits different from $x\mathtt{55}w$, which may be at the positions $\ell+1$, $\ell+2$, $\ell+3$ and $\ell+4$.

Assume for the sake of contradiction that $|w| \neq |w'|$. Since w and w' are both of odd length, the positions of $\mathtt{5}$ in the strings $x\mathtt{55}w$ and $x'\mathtt{55}w'$ are different. Hence $x\mathtt{55}w$ and $x'\mathtt{55}w'$ differ at exactly four positions, which are the positions of $\mathtt{5}$ in them.

Note that if four digits are modified by adding y, then the digit in the position $\ell+4$ can only be incremented by 1 due to a carry from the previous position. Since one of the strings $x\mathtt{55}w$, $x'\mathtt{55}w'$ has the digit $\mathtt{5}$ in the position $\ell+4$, the other string should have a digit $\mathtt{4}$ or $\mathtt{6}$ in the same position. Because the latter digits are not encodings of any symbols, this yields a contradiction. □

The next statement describes the operation of Carry: when applied to a configuration $x\mathtt{55}w$ with the counter x having a single carry digit $\mathtt{2}$, Carry changes this digit to $\mathtt{0}$ and increments the next digit, turning it to $\mathtt{1}$ or $\mathtt{2}$. The tape contents is not altered, only the carry digit is propagated to the next higher position. Note that all operations are in $|\Sigma|$-ary notation.

Lemma 3 (Carry propagation) *For every* $x \in \mathrm{Counter}'$ *and for every* $w \in \mathrm{Tape}$, $\mathrm{Carry}(x\mathtt{55}w) = \{x'\mathtt{55}w\}$ *for some* $x' \in \mathrm{Counter} \cup \mathrm{Counter}'$ *with* $\mathrm{Value}(x') = \mathrm{Value}(x)$.

For every string $\alpha \in \Sigma^*$ *of any different form,* $\mathrm{Carry}(\alpha) = \varnothing$.

Proof The inner intersection with $\{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape}$ ensures that the set $\mathrm{Carry}(\alpha)$ is non-empty only for $\alpha = x\mathtt{55}w$ with $x \in \mathrm{Counter}'$ and $w \in \mathrm{Tape}$.

The goal is to prove that if $x = \mathtt{2}\widetilde{x} \in \mathrm{Counter}'$ and $w \in \mathrm{Tape}$, then

$$\mathrm{Carry}(\mathtt{2}\widetilde{x}\mathtt{55}w) = \{\mathtt{10}\widetilde{x}\mathtt{55}w\},$$

and if $x = \widehat{x}c\mathtt{2}\widetilde{x} \in \mathrm{Counter}'$ and $w \in \mathrm{Tape}$, then

$$\mathrm{Carry}(\widehat{x}c\mathtt{2}\widetilde{x}\mathtt{55}w) = \{\widehat{x}(c+1)\mathtt{0}\widetilde{x}\mathtt{55}w\}.$$

If a string $x\mathtt{55}w$, with $x \in \mathrm{Counter}' \cup \mathrm{Counter}$ and $w \in \mathrm{Tape}$, is substituted into the expression Carry, then the first subexpression produces all strings of the form

$$u \in (x\mathtt{55}w + \mathtt{10}^*) \cap \{\mathtt{0},\mathtt{1}\}^*\mathtt{3}\{\mathtt{0},\mathtt{1}\}^*\,\mathtt{55}\,\mathrm{Tape}\,.$$

Consider the possible changes made to $x\mathtt{55}w$ to obtain u. As $\mathtt{1}$ is added only to one digit, there cannot be a carry, because the digit $|\Sigma|-1$ is not used for encoding. Therefore, only one digit is modified in $x\mathtt{55}w$. Since $x\mathtt{55}w$ does not contain the digit $\mathtt{3}$ that occurs in u, the unique digit $\mathtt{2}$ in x must be replaced by $\mathtt{3}$. Denote $u = \widetilde{x}\mathtt{55}w$.

Consider the string u' (any such string if it is not unique) obtained in the next subexpression:

$$u' \in \left(u + (\mathtt{10} \boxminus \mathtt{3})\mathtt{0}^*\right) \cap \left(\{\mathtt{0},\mathtt{1}\}^+ \cup \{\mathtt{0},\mathtt{1}\}^*\mathtt{2}\{\mathtt{0},\mathtt{1}\}^*\right)\mathtt{55}\,\mathrm{Tape}\,.$$

Let $u' = u + y$, with $y \in (\mathtt{10} \boxminus \mathtt{3})\mathtt{0}^* = (|\Sigma|-3)\mathtt{0}^*$. By Lemma 2, $u' = x'\mathtt{55}w'$ and $|w'| = |w|$.

Consider the changes in $x'\mathtt{55}w'$ as compared to $\widetilde{x}\mathtt{55}w$. Since there is a digit $\mathtt{3}$ in $\widetilde{x}$ and there is no such digit in x', the position of $\mathtt{3}$ in $\widetilde{x}$ is one of the modified positions. Denote the number of this position by k. Because the addition of y has modified the digit $\mathtt{3}$, this means that the unique non-zero digit in y is in position k or $k-1$. If it is in the position $k-1$, then the digit $\mathtt{3}$ can only be modified by adding 1 as a carry from the position $k-1$. This cannot be the case, as the digit $\mathtt{4}$ is not used in the encoding. Therefore, the non-zero digit in y is in the position k; then adding y to $\widetilde{x}\mathtt{55}w$ replaces $\mathtt{3}$ with $\mathtt{0}$ and results in a carry, thus increasing the digit in the position $k+1$ by 1. Note that, in particular, no changes were made to w, and hence $w' = w$.

Finally, consider the values of the counters x and x'. The value of x is $\sum c_i 2^i$, where c_i is the digit in the i-th position. If x has no digit in the position $k+1$, then assume for the purposes of calculation that $c_{k+1} = 0$ (this does not influence the value of the counter). In x', the digit $\mathtt{2}$ was replaced by $\mathtt{0}$, hence $c'_k = \mathtt{0}$. In the position $k+1$, the digit c_{k+1} was replaced with $c_{k+1}+1$. If there was no actual digit c_{k+1} in x, then a new digit $c'_{k+1} = \mathtt{1}$ has been created. In any case x' contains the digit $c'_{k+1} = c_{k+1}+1$ in this position. All other digits of the counters are left intact. Then the difference of the values of the counters is determined by the positions k and $k+1$, and

$$\mathrm{Value}(x) - \mathrm{Value}(x') = (c_{k+1} \cdot 2^{k+1} + \mathtt{2} \cdot 2^k) - ((c_{k+1}+1) \cdot 2^{k+1} + \mathtt{0} \cdot 2^k) = 0,$$

that is, the value of the counter has been preserved. □

According to Lemma 3, Carry basically moves the carry higher by one position. The next lemma shows that sufficiently many iterations of Carry always eliminate the carry digit: given a counter with the notation $x' = \widetilde{x}\mathtt{01}^{k-1}\mathtt{2} \in \mathrm{Counter}'$, Carry^k transforms it to $x = \widetilde{x}\mathtt{10}^{k-1}\mathtt{0} \in \mathrm{Counter}$.

Lemma 4 (Termination of carry propagation) *For every* $x' \in \mathrm{Counter} \cup \mathrm{Counter}'$ *and* $w \in \mathrm{Tape}$ *there exist* $x \in \mathrm{Counter}$ *and a number* $k \geqslant 0$*, such that* $\mathrm{Carry}^k(x'\mathtt{55}w) = \{x\mathtt{55}w\}$ *and* $\mathrm{Value}(x) = \mathrm{Value}(x')$.

Proof If $x' \in \mathrm{Counter}$, then statement of the lemma is satisfied for $k = 0$ and $x = x'$.

Let $x' \in \mathrm{Counter}'$ and construct a sequence $x_0, x_1, \ldots, x_k$, with $x_i \in \mathrm{Counter}'$ and $\mathrm{Value}(x_i) = \mathrm{Value}(x')$, where k shall be determined below, as follows. Let $x_0 = x'$. For every $i \geqslant 1$, consider $\mathrm{Carry}(x_{i-1}\mathtt{55}w)$, which, by Lemma 3, equals $\{x_i\mathtt{55}w\}$ for some $x_i \in \mathrm{Counter}' \cup \mathrm{Counter}$ with $\mathrm{Value}(x_i) = \mathrm{Value}(x_{i-1})$. If $x_i \in \mathrm{Counter}$, then $k = i$ and $x = x_i$ satisfy the statement of the lemma. Otherwise, if $x_i \in \mathrm{Counter}'$, then the construction of the sequence continues.

Note that the numerical value of each configuration $x_{i+1}\mathtt{55}w$ (as a number in base-$|\Sigma|$ notation) is strictly greater than in $x_i\mathtt{55}w$, and hence all elements of the sequence are distinct. Since there exist only finitely many elements of $\mathrm{Counter}'$ with the same value, the sequence cannot be infinite and eventually $x_i \in \mathrm{Counter}$ is obtained. □

The next lemma determines the operation of Jump, which can be described as follows. This function is applicable to configurations in which the head scans the first symbol, and the result of Jump on every such configuration is the *previous* configuration, in which the head is at the right-most position beyond the end of the string, while the value of the counter x is greater by 1.

Lemma 5 *Let* $x = \widetilde{x}c \in \mathrm{Counter}$ *with* $c \in \{\mathtt{0}, \mathtt{1}\}$ *and let* $w = \langle q\rangle\widetilde{w}\mathtt{0} \in \mathrm{Tape}$ *with* $q \in Q$*, that is,* w *is a configuration with the head over the first symbol. Then* $\mathrm{Jump}(x\mathtt{55}w) = \{\widetilde{x}(c+1)\mathtt{550}\widetilde{w}\langle q\rangle\}$.

For any string $\alpha \in \Sigma^*$ *of a different form,* $\mathrm{Jump}(\alpha) = \varnothing$.

Proof The inner subexpression of $\mathrm{Jump}(x\mathtt{55}w)$,

$$\{x\mathtt{55}w\} \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_q^{\ell},$$

ensures that $w = \langle q\rangle\widetilde{w}\mathtt{0}$ for some $\widetilde{w} \in \langle \Gamma\rangle(\mathtt{0}\langle\Gamma\rangle)^*$, that is, that the digit specifying the state of the machine is in the left-most position. If w is of a different form, then

$\mathrm{Jump}(x\mathtt{55}w) = \varnothing$. Fix an arbitrary state $q \in Q$; as the outermost operation in Jump, a union over all q will be taken.

The next subexpression performs an addition

$$x\mathtt{55}\langle q\rangle\widetilde{w}\mathtt{0} + (\mathtt{1000} \boxminus \langle q\rangle)(\mathtt{00})^{+} + \langle q\rangle,$$

which is meant to remove q from the beginning of the tape, increment the counter and place q in the end of the tape. Consider an arbitrary $y = (\mathtt{1000} \boxminus \langle q\rangle)(\mathtt{00})^{k} + \langle q\rangle$, with $k \geqslant 1$. Denote $u = x\mathtt{55}w + y$ and assume that

$$u \in (\mathrm{Counter}' \cup \mathrm{Counter})\mathtt{55}\,\mathrm{Tape}_q,$$

as the subsequent operation in Jump is an intersection with this set. Let $u = x'\mathtt{55}w'$ with $x' \in \mathrm{Counter} \cup \mathrm{Counter}'$ and $w' \in \mathrm{Tape}$. Also note that $w' = \mathtt{0}\widetilde{w}'\langle q\rangle$, as the right-most digit in y is $\langle q\rangle$ and the right-most digit in w is $\mathtt{0}$, and there is only one digit from $\langle Q\rangle$ in w'.

Since y has non-zero digits only in the positions $2k+1, 2k+2, 2k+3$ and 1, and the digit $|\Sigma| - 1$ does not encode any symbol, adding y cannot change any digit in $x\mathtt{55}w$ in positions higher than $2k+4$. Let $2\ell = |\widetilde{w}\mathtt{0}|$. Then adding y to w modifies the digit in the position $2\ell + 1$, which is $\langle q\rangle$. Hence, $2\ell + 1 = 2k + 1$ or $2\ell + 1 = 2k + 3$.

If $2\ell+1 = 2k+3$, then there is either $\langle q\rangle$ or $\langle q\rangle - 1$ in the position $2\ell+1$ in $x\mathtt{55}w+y$. Moreover, the digit $\mathtt{5}$ in the position $2\ell + 3 = 2k + 5$ in $x\mathtt{55}w$ was not modified by adding y. Since the position $2\ell + 1$ is to the right of $\mathtt{55}$ in $x'\mathtt{55}w'$, it contains $\mathtt{0}$. This is a contradiction, as $\langle q\rangle > \langle q\rangle - 1 > \mathtt{0}$.

Hence, $2\ell+1 = 2k+1$. Let $x = \widetilde{x}c$. Then $x\mathtt{55}w+y = \widetilde{x}(c+1)\mathtt{550}\widetilde{w}\langle q\rangle$, and therefore $x' = \widetilde{x}(c+1)$ and $w' = \mathtt{0}\widetilde{w}\langle q\rangle$, as stated in the lemma. □

The next operation is Move, which represents symbol manipulation, head movement and state change of a Turing machine according to the transitions specified in δ. Generally, when $\mathrm{Move}_{q',a',q,a}$ is applied to a valid configuration, it computes the *preceding configuration* of the machine. This configuration is unique because of the restriction built in $\mathrm{Move}_{q',a',q,a}$: the intersections therein ensure that in the current configuration the machine is in the state q' and the symbol to the left rewritten at the previous step is a', while in the previous configuration the machine was in the state q and used to scan the symbol a. For all other configurations and in all other cases, the function produces the empty set.

Lemma 6 *Let $q, q' \in Q$ with $q \neq q'$, and let $a, a' \in \Gamma$. Let $x \in \mathrm{Counter}$ and $w = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w} \in \mathrm{Tape}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widetilde{w} \in (\langle\Gamma\rangle\mathtt{0})^*$. Then $\mathrm{Move}_{q',a',q,a}(x\mathtt{55}w) = \{x\mathtt{55}\widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}\}$.*

For every string $\alpha \in \Sigma^*$ of a different form, $\mathrm{Move}_{q',a',q,a}(\alpha) = \varnothing$.

Proof Fix a', q', a and q. The inner subexpression of $\mathrm{Move}_{q',a',q,a}$,

$$x\mathtt{55}w \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{a'q'},$$

ensures that $w = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widetilde{w} \in (\langle\Gamma\rangle\mathtt{0})^*$. For any w of a different form, $\mathrm{Move}_{q',a',q,a}(x\mathtt{55}w)$ is empty.

The next subexpression performs the operation

$$x\mathtt{55}w + \big(\langle q\rangle\langle a\rangle\mathtt{0} \boxminus \langle a'\rangle\langle q'\rangle\big)(\mathtt{00})^* \cap \mathrm{Counter}\,\mathtt{55}\,\mathrm{Tape}_{qa},$$

which is designed to replace the digits $\mathtt{0}\langle a'\rangle\langle q'\rangle$ in w with the digits $\langle q\rangle\langle a\rangle\mathtt{0}$. The task is to show that the addition always proceeds according to this plan.

Let

$$y = \big(\langle q\rangle\langle a\rangle\mathtt{0} \boxminus \langle a'\rangle\langle q'\rangle\big)\mathtt{0}^{2k} \in \big(\langle q\rangle\langle a\rangle\mathtt{0} \boxminus \langle a'\rangle\langle q'\rangle\big)(\mathtt{00})^*$$

and consider the string $x'\mathtt{55}w' = x\mathtt{55}w + y$, where $x' \in \text{Counter}$ and $w' \in \text{Tape}$. By Lemma 2, $|w'| = |w|$.

As y has non-zero digits only in positions $2k+1, 2k+2, 2k+3$, while the digit $|\Sigma|-1$ is not a valid encoding of any symbol, adding y cannot change any digits in $x\mathtt{55}w$ in positions higher than $2k+4$.

Let $2\ell = |\widetilde{w}|$. Since $q \neq q'$, w and w' must differ in the position $2\ell+1$, where w has the digit $\langle q'\rangle$. Therefore, $2\ell+1 = 2k+3$ or $2\ell+1 = 2k+1$.

Suppose $2\ell+1 = 2k+3$, that is, the digit $\langle q'\rangle$ in w is added to $\langle q\rangle$ or $\langle q\rangle - 1$ in y, with a possible carry from the lower digits. Then $x\mathtt{55}w+y$ has a digit $(\langle q\rangle + \langle q'\rangle - 1)$, $(\langle q\rangle + \langle q'\rangle)$ or $(\langle q\rangle + \langle q'\rangle + 1)$ (modulo $|\Sigma|$ in each case) in the position $2\ell+1$. Since $\langle q\rangle, \langle q'\rangle \leqslant 6 + |Q|$ and $q \neq q'$, it follows that $\langle q\rangle + \langle q'\rangle \leqslant 11 + 2|Q|$ and $\langle q\rangle + \langle q'\rangle + 1 \leqslant 12 + 2|Q|$. Each sum is smaller than $|\Sigma|$ and is therefore represented by a single digit. However, each of these digits is greater than $\langle q\rangle$, and hence all of them are filtered out by the intersection with $\text{Counter}\,\mathtt{55}\,\text{Tape}_{qa}$.

In the other case of $2\ell+1 = 2k+1$, the addition proceeds as expected, and $x' = x$ and $w' = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$, as stated in the lemma. □

The flow control of an alternating Turing machine includes existential and universal nondeterminism in the corresponding states, and a single step is in fact a disjunction or conjunction of several transitions as specified in Move. This logic is transcribed in the expression $\text{Step}(X)$, which computes the set of all *previous configurations*, from which machines in a universal state make all their transitions to configurations in X and machines in an existential state make at least one of their transitions to some configuration in X. This implements one step of the computation of the machine, backwards.

Lemma 7 *Let $x \in \text{Counter}$ and $w \in \text{Tape}$, let $q \in Q$ be the state encoded in w. Let $X \subseteq \mathbb{N}$. Then $x\mathtt{55}w \in \text{Step}(X)$ if and only if the following conditions hold:*

- *the configuration w has the head **not** in the position beyond the right-most symbol, that is, $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widetilde{w} \in (\langle\Gamma\rangle\mathtt{0})^*$ and $a \in \Gamma$;*
- *if $q \in Q_E$, then $x\mathtt{55}w' \in X$ for some configuration w' among successors to w;*
- *if $q \in Q_A$, then $x\mathtt{55}w' \in X$ for every configuration w' among successors to w.*

Proof $\Leftrightarrow$ Consider the definition of Step:

$$\begin{aligned}&\text{Step}(X) = \\ &= \Big(\bigcup_{\substack{\widehat{q}\in Q_E\\ \widehat{a}\in\Gamma}} \bigcup_{(q',a')\in\delta(\widehat{q},\widehat{a})} \text{Move}_{q',a',\widehat{q},\widehat{a}}(X)\Big) \cup \Big(\bigcup_{\substack{\widehat{q}\in Q_A\\ \widehat{a}\in\Gamma}} \bigcap_{(q',a')\in\delta(\widehat{q},\widehat{a})} \text{Move}_{q',a',\widehat{q},\widehat{a}}(X)\Big).\end{aligned}$$

Assume that $x\mathtt{55}w \in \text{Step}(X)$, let $a \in \Gamma$ be the symbol scanned by the head of the machine in the configuration w, and let $q \in Q$ be the current state. Then, according to Lemma 6, $x\mathtt{55}w \in \text{Move}_{q',a',\widehat{q},\widehat{a}}(X)$ only if $(\widehat{q},\widehat{a}) = (q,a)$, and hence the subexpressions $\text{Move}_{q',a',\widehat{q},\widehat{a}}(X)$ with $(\widehat{q},\widehat{a}) \neq (q,a)$ need not be taken into account.

First suppose that q is an existential state. Then

$$x\mathtt{55}w \in \bigcup_{(q',a')\in\delta(q,a)} \mathrm{Move}_{q',a',q,a}(X),$$

that is, there exist $q' \in Q$ and $a' \in \Gamma$ with $x\mathtt{55}w \in \mathrm{Move}_{q',a',q,a}(X)$ for some $(q',a') \in \delta(q,a)$. Note that $q \neq q'$ by the technical assumption that the machine changes its state upon every transition. Since $\mathrm{Move}_{q',a',q,a}$ is distributive over infinite union by Lemma 1, there exists a number $n \in X$ with $x\mathtt{55}w \in \mathrm{Move}_{q',a',q,a}(n)$. Then, by Lemma 6, n must be of the form $x\mathtt{55}w'$ with $w' = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$ for some $\widehat{w} \in (\mathtt{0}\langle\Gamma\rangle)^*$ and $\widehat{w} \in (\langle\Gamma\rangle\mathtt{0})^*$, and with $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$. Since $(q',a') \in \delta(q,a)$, w' is a successor configuration to w, and $x\mathtt{55}w' \in X$. The position of the head in w is to the left of the right-most symbol.

The case of $q \in Q_A$ is similar. It follows from $x\mathtt{55}w \in \mathrm{Step}(X)$ that

$$x\mathtt{55}w \in \bigcap_{(q',a')\in\delta(q,a)} \mathrm{Move}_{q',a',q,a}(X),$$

that is, for all $q' \in Q$ and $a' \in \Gamma$ with $(q',a') \in \delta(q,a)$ it holds that $x\mathtt{55}w \in \mathrm{Move}_{q',a',q,a}(X)$. As in the previous case, this implies that $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$ and there is $x\mathtt{55}w'_{q',a'} \in X$ with $w'_{q',a'} = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$. These are consecutive configurations, and every successor configuration to w is of this form for some $(q',a') \in \delta(q,a)$. Then the required element $x\mathtt{55}\widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$ is in X for all q' and a' with $(q',a') \in \delta(q,a)$. Also note that $\delta(q,a) \neq \varnothing$ by assumption, and hence there is at least one such pair (q',a'). Hence, w is of the required form with the head not beyond the right-most symbol.

$\Leftarrow$ Let $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$ and first consider the case of $q \in Q_E$. Let w' be one of the next configurations of the machine with $x\mathtt{55}w' \in X$. Then $w' = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$ for some $(q',a') \in \delta(q,a)$, and it is known that $q \neq q'$. By Lemma 6, $\mathrm{Move}_{q',a',q,a}(x\mathtt{55}w') = \{x\mathtt{55}w\}$. Since $\mathrm{Move}_{q',a',q,a}(x\mathtt{55}w') \subseteq \mathrm{Step}(X)$, this shows that $x\mathtt{55}w \in \mathrm{Step}(X)$.

If $q \in Q_A$, then, by assumption, $x\mathtt{55}w' \in X$ for all configurations w' immediately following w. That is, for all $(q',a') \in \delta(q,a)$, $x\mathtt{55}w'_{q',a'} \in X$, where $w'_{q',a'} = \widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w}$. For every such pair, by Lemma 6, $x\mathtt{55}w \in \mathrm{Move}_{q',a',q,a}(x\mathtt{55}w'_{q',a'})$. Hence,

$$x\mathtt{55}w \in \bigcap_{(q',a')\in\delta(q,a)} \mathrm{Move}_{q',a',q,a}(X),$$

and therefore $x\mathtt{55}w \in \mathrm{Step}(X)$. □

Thus the formal meaning of all auxiliary operations has been established, and the equations can now be analyzed. The equation for X states that a configuration leads to acceptance if and only if it is itself accepting (Final), or one can directly proceed from it to a configuration leading to acceptance ($\mathrm{Step}(X)$), or that it is a configuration obtained in Y. The equation for Y specifies circular rotation of the tape by $\mathrm{Jump}(X)$ and implements iterated carry propagation as in Lemma 4 by a self-reference $\mathrm{Carry}(Y)$. Altogether, the least solution of these equations corresponds to the computation of the machine as follows:

Lemma 8 *Let (L_X, L_Y) be the least solution of the system* (3).

I. Let $x \in$ Counter, $w \in$ Tape and $x\mathtt{55}w \in L_X$. Then M accepts starting from the configuration w.

II. Conversely, if M accepts starting from a configuration $w \in \mathrm{Tape}$, and the longest path in the tree of the accepting computation has length ℓ, then $x\mathtt{55}w \in L_X$ for each $x \in \mathrm{Counter}$ with $\mathrm{Value}(x) \geqslant \ell$.

Proof As the least solution of the system is computed by fixpoint iteration (1), denote by $L_X^{(k)}$ and $L_Y^{(k)}$ the X- and Y-components of the vector $\varphi^k(\varnothing, \ldots, \varnothing)$ obtained after $k \geqslant 0$ iterations. Then $x\mathtt{55}w \in L_X$ if and only if $x\mathtt{55}w \in L_X^{(k)}$ for some $k \geqslant 1$.

(I) Assume that $x\mathtt{55}w \in L_X^{(k)}$. It has to be proved that the Turing machine accepts starting from the configuration w. The proof is an induction on k.

By the equation for X, $x\mathtt{55}w \in L_X^{(k)}$ means that either $x\mathtt{55}w \in \mathrm{Final}$, or $x\mathtt{55}w \in \mathrm{Step}(L_X^{(k-1)})$, or $x\mathtt{55}w \in L_Y^{(k-1)}$. If $x\mathtt{55}w \in \mathrm{Final}$, then w is an accepting configuration, as the Turing machine is already in an accepting state. Consider the other two cases.

Let $x\mathtt{55}w \in \mathrm{Step}(L_X^{(k-1)})$, and let $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$; the configuration is of this form by Lemma 7. Consider the set of numbers $S = \{x\mathtt{55}\widehat{w}\mathtt{0}\langle a'\rangle\langle q'\rangle\widetilde{w} \mid (q', a') \in \delta(q, a)\}$ representing all possible next configurations of the machine. Suppose first that $q \in Q_A$. Then, by Lemma 7, all numbers in S are in $L_X^{(k-1)}$, and by the induction hypothesis, all numbers in $L_X^{(k-1)}$ represent configurations from which the machine accepts. Hence the machine accepts starting from all successor configurations to w, and then, by definition, it accepts starting from w.

The case of $q \in Q_E$ is treated similarly. Again, by Lemma 7, at least one number from S is in $L_X^{(k-1)}$, and every number in $L_X^{(k-1)}$ represents a configuration from which the machine accepts, by the induction hypothesis. Accordingly, the machine accepts starting from the configuration w, because it accepts starting from one of its successor configurations.

Consider the other case of $x\mathtt{55}w \in L_Y^{(k-1)}$, that is, of $x\mathtt{55}w$ obtained by processing the carry in the counter. This processing may be reconstructed as a finite sequence x_{k-1}, x_{k-2}, …, $x_{k_0} \in \mathrm{Counter} \cup \mathrm{Counter}'$, where the number $k_0 \geqslant 0$ is determined later, and, for all $i \in \{k-1, k-2, \ldots, k_0\}$,

$$\begin{aligned} &\mathrm{Value}(x_i) = \mathrm{Value}(x), \\ &x_i\mathtt{55}w \in L_Y^{(i)}, \\ &x_i\mathtt{55}w = \mathrm{Carry}(x_{i-1}\mathtt{55}w) \quad (\text{unless } i = k_0). \end{aligned}$$

Let $x_{k-1} = x$. Each string of digits x_i for $i = k-2, k-3, \ldots$ is defined by a backward induction as follows.

Assume that $x_i\mathtt{55}w \in L_Y^{(i)}$, and hence $x_i\mathtt{55}w \in \mathrm{Carry}(L_Y^{(i-1)})$ or $x_i\mathtt{55}w \in \mathrm{Jump}(L_X^{(i-1)})$. In the former case, by Lemma 1, there exists a number $n \in L_Y^{(i-1)}$ with $x_i\mathtt{55}w \in \mathrm{Carry}(n)$. Then, by Lemma 3, n must be of the form $x'\mathtt{55}w$ for some x' with $\mathrm{Value}(x') = \mathrm{Value}(x_i)$, and it holds that $\mathrm{Carry}(x'\mathtt{55}w) = \{x_i\mathtt{55}w\}$. Then $x_{i-1} = x'$ forms the next element of the sequence.

In the latter case, $x_i\mathtt{55}w \in \mathrm{Jump}(L_X^{(i-1)})$. Again, Lemma 1 implies that there is a number $n \in L_X^{(i-1)}$ with $x_i\mathtt{55}w \in \mathrm{Jump}(n)$. Then, according to Lemma 5, $n = x'\mathtt{55}w'$, where $x' \in \mathrm{Counter}'$ with $\mathrm{Value}(x') = \mathrm{Value}(x_i) - 1 = \mathrm{Value}(x) - 1$, $w' = \langle q\rangle\widetilde{w}\mathtt{0}$ and $w = \mathtt{0}\widetilde{w}\langle q\rangle$; that is, $\mathrm{Jump}(x'\mathtt{55}w') = \{x_i\mathtt{55}w\}$. Then k_0 is defined as i, which completes the construction of the sequence.

It has been shown that there exists $x'\mathtt{55}w' \in L_X^{(k_0-1)}$ with $\mathrm{Value}\, x' = \mathrm{Value}\, x - 1$, such that the machine goes from the configuration w to the configuration w'. By

the induction hypothesis for $x'\mathtt{55}w'$, the machine accepts from the configuration w'. Therefore, the machine accepts starting from w, as claimed.

It is left to mention that the case of $x_i\mathtt{55}w \in \mathrm{Jump}(L_X^{(i-1)})$ in the above proof eventually occurs, because otherwise the sequence would continue until $x_0\mathtt{55}w \in L_Y^{(0)} = \varnothing$, which is impossible.

(II) For the converse statement, let w be a configuration, and assume that there is an accepting computation starting from w, with the longest path of length ℓ. The claim is that $x\mathtt{55}w \in L_X$ holds for every $x \in \mathrm{Counter}$ with the value at least ℓ. This is proved by induction on ℓ.

If $\ell = 0$, then w is a configuration in the accepting state, and therefore, by the equation for X in the system, $x\mathtt{55}w \in \mathrm{Final} \subseteq L_X$ for all $x \in \mathrm{Counter}$.

Assume there is an accepting computation starting from w with the longest path of length $\ell + 1$. Suppose first that $w = \widehat{w}\langle q\rangle\langle a\rangle\mathtt{0}\widetilde{w}$ with $\widehat{w}, \widetilde{w} \in \Sigma^*$, $a \in \Gamma$ and $q \in Q$, that is, the configuration w has the head anywhere except in the position beyond the right-most symbol. Consider the case of $q \in Q_A$. Then the machine accepts from each successor configuration to w, and longest path in each of these accepting computations is of length at most ℓ. Hence all strings of the form $x\mathtt{55}w'$, where w' is a successor configuration to w and $x \in \mathrm{Counter}$ represents a counter of value at least ℓ, are in L_X by the induction hypothesis. Then, by Lemma 7, $x\mathtt{55}w \in \mathrm{Step}(L_X)$. By the equation for X, $x\mathtt{55}w \in L_X$, which proves this case.

Now consider the case when $q \in Q_E$. Fix any $x \in \mathrm{Counter}$ of value at least ℓ. At least for one successor configuration to w, the Turing machine accepts starting from it, with the longest path of length at most ℓ. Accordingly, at least one string of the form $x\mathtt{55}w'$, where w' is one of the successor configurations to w, is in L_X by the induction hypothesis. Therefore, by Lemma 7, $x\mathtt{55}w \in \mathrm{Step}(L_X)$, and, by the equation for X, $x\mathtt{55}w \in L_X$, as stated in the lemma.

Finally, consider the case where the head of the Turing machine is in the position beyond the right-most symbol, and let $w = \mathtt{0}\widetilde{w}\langle q\rangle$. Let $w' = \langle q\rangle\widetilde{w}\mathtt{0}$ be the next configuration, from which the machine accepts with the longest path of length ℓ. Let $x' \in \mathrm{Counter}$ be a counter of value at least ℓ. By the induction hypothesis, $x'\mathtt{55}w' \in L_X$. Then, by Lemma 5, there is a string $x''\mathtt{55}w \in \mathrm{Jump}(L_X) \subseteq L_Y$, where $x'' \in \mathrm{Counter}' \cup \mathrm{Counter}$ and $\mathrm{Value}(x'') = \mathrm{Value}(x') + 1$. Hence, by Lemma 4, there exists $k \geqslant 0$, for which $x\mathtt{55}w \in \mathrm{Carry}^k(L_Y)$, where x is the unique element of Counter with $\mathrm{Value}(x'') = \mathrm{Value}(x)$. By the equation for Y in the system, $\mathrm{Carry}(L_Y) \subseteq L_Y$, and since Carry is monotone, this implies the following chain of inclusions:

$$\mathrm{Carry}^k(L_Y) \subseteq \mathrm{Carry}^{k-1}(L_Y) \subseteq \mathrm{Carry}^{k-2}(L_Y) \subseteq \ldots \subseteq \mathrm{Carry}(L_Y) \subseteq L_Y.$$

Therefore, $x\mathtt{55}w \in L_Y$, which, by the equation for X in the system (3), implies $x\mathtt{55}w \in L_X$, as claimed. □

It remains to observe that the number of steps made by the machine is exponentially bounded, and hence the acceptance of a string by the machine is represented by the following number in the least solution of the constructed system:

Main Claim The ATM M accepts a string $a_1 \ldots a_n \in \Omega^+$ if and only if

$$\mathtt{1}\mathtt{0}^{n\log(|\Gamma|)+\log(n+1)+\log(|Q|)}\mathtt{55}\langle q_0\rangle\langle a_1\rangle\mathtt{0}\langle a_1\rangle\mathtt{0}\ldots\langle a_n\rangle\mathtt{0} \in L_X.$$

Proof The initial configuration of M on $a_1 \dots a_n$ is represented by the sequence of digits $w = \langle q_0 \rangle \langle a_1 \rangle \mathtt{0} \langle a_2 \rangle \mathtt{0} \dots \langle a_n \rangle \mathtt{0} \in \text{Tape}$.

$\Rightarrow$ If M accepts starting from this configuration, then the longest path in the accepting computation consists of at most

$$(n+1) \cdot |Q| \cdot |\Gamma|^n \leqslant 2^{\log(n+1)} \cdot 2^{\log|Q| + n \log|\Gamma|}$$

steps, since all configurations forming this path must be different. Then, by Lemma 8, for $x = \mathtt{1}^{\log(n+1)+n\log|\Gamma|+\log|Q|}$ with $\text{Value}(x) = 2^{\log(n+1)+n\log|\Gamma|+\log|Q|}$ it holds that $x\mathtt{55}w \in L_X$.

$\Leftarrow$ Conversely, if there exists $x \in \text{Counter}$ with $x\mathtt{55}w \in L_X$, then, according to Lemma 8, M accepts starting from the configuration w. □

Proof (Proof of Theorem 2) The system of equations constructed above has an EXPTIME-hard least solution. It uses constant sets of numbers with a regular base-$|\Sigma|$ notation, which are expressed in additional equations for additional variables constructed according to Theorem 1.

To see that the least solution of every system is in EXPTIME, it is sufficient to represent it as a conjunctive grammar over a unary alphabet. Then, given a number n, its membership in the least solution can be tested by supplying the string a^n to a known cubic-time parsing algorithm for conjunctive grammars [17]. Its time is cubic in n, hence exponential in the length of the binary notation of n. □

This establishes the computational complexity of sets of numbers specified by resolved systems of equations with union, intersection and addition, which is the main result of this paper.

4 The membership problem

Consider the *general membership problem* for these equations, stated as follows: "Given a system $X_i = \varphi_i(X_1, \dots, X_m)$ and given a number n *in binary notation*, determine whether n is in the first component of the least solution of the system".

Theorem 3 *The general membership problem for resolved systems of equations over sets of numbers with the operations of union, intersection and addition is EXPTIME-complete.*

Proof **Membership in EXPTIME.** The existence of such an algorithm can be inferred from the known polynomial-time algorithm for solving the membership problem for conjunctive grammars [18]. It is sufficient to represent the given system as a conjunctive grammar over a unary alphabet, with a linearly bounded blow-up, and then represent the given number n as a string a^n, with an exponential blow-up.

An exponential-time algorithm for equations over sets of numbers can be constructed directly as follows. Given a number n and a resolved system with m variables, the algorithm will simulate fixpoint iteration as in (1), but all sets will be computed as subsets of $\{0, \dots, n\}$. The algorithm thus uses variables $X_i \subseteq \{0, \dots, n\}$, which are initially empty, and which are updated at every step by substituting their values into the right-hand side of the system. Up to $m(n+1)$ such iterations can be done until the sets stabilize, when the algorithm can answer whether n is in X_1. Each iteration

		Representable sets	Membership problem	Equivalence problem
$\{\cup,+\}$	expressions	Finite	NP-complete [24]	Π_2^P-complete [6]
	circuits	Finite	NP-complete [7, 15]	Π_2^P-complete [6]
	equations	Ult. periodic [5]	NP-complete [7, 22]	?
$\{\cup,\cap,+\}$	expressions	Finite	PSPACE-complete [15]	Π_2^P-complete [6]
	circuits	Finite	PSPACE-complete [15]	PSPACE-complete [6]
	equations	$\subsetneq$ **EXPTIME**	**EXPTIME-complete**	Π_1^0-complete [9]

Table 1 Comparison of formalisms over sets of integers.

is polynomial in $n+m$, and so is the entire algorithm. Since n is given to the algorithm in binary notation, the size of the instance of the general membership problem is $\log n + m$, and hence the algorithm makes at most exponentially many iterations each working in exponential time.

The **EXPTIME-hardness** of the general membership problem immediately follows from Theorem 2 by fixing the system of equations. □

Recalling that *equations* over sets of numbers are a generalization of *circuits* and *expressions* over sets of numbers, Theorem 3 can be directly compared to the existing results on the complexity of these formalisms. If the allowed operations are addition and union only, then, according to Stockmeyer and Meyer [24, Thm. 5.1], the membership problem is NP-hard for expressions. At the same time, for equations with these operations it can be solved by an NP algorithm due to Huynh [7] (or by a more specialized NP algorithm of Plandowski and Rytter [22, Thm.8]), and hence the problem is NP-complete for all three models. Once the operation of intersection is added, the complexity increases: McKenzie and Wagner [15] showed the problem to be PSPACE-complete for expressions and circuits alike, and Theorem 3 states its EXPTIME-completeness for equations.

These results are summarized in the middle column of Table 1. The left column of the table characterizes the families of sets representable by each of these six formalisms. Obviously, expressions and circuits can represent only finite sets, as any Boolean combinations and sums of finite sets are finite. The sets represented by equations with union and addition are bound to be ultimately periodic, because all context-free languages over a unary alphabet are regular [5]. The contribution of this paper is that equations with union, intersection and addition can represent some EXPTIME-complete sets. At the same time, these equations cannot represent the whole class $\mathrm{EXPTIME} = \bigcup_{k \geqslant 1} \mathrm{DTIME}(2^{n^k})$, because their solutions lie in $\mathrm{DTIME}(2^{n^2})$, which is a proper subset of EXPTIME due to the time hierarchy theorem. This case stands out of the rest of the formalisms in Table 1, as these equations are not only able to represent non-periodic sets, but can actually represent sets as hard as the general membership problem for this family.

Another important decision problem is the *equivalence problem*, which constitutes testing whether two given systems (expressions, circuits, etc.) define the same set. For expressions and circuits over sets of numbers this problem has been studied by Glaßer et al. [6], who proved, in particular, that testing equivalence of expressions or circuits with union and addition is Π_2^P-complete, and if intersection is also allowed, then the problem remains Π_2^P-complete for expressions but becomes PSPACE-complete for circuits. To compare, the equivalence problem for equations over sets of numbers with

union, intersection and addition was proved to be undecidable by the authors [9]; to be precise, it is Π_1^0-complete, and it remains Π_1^0-complete even if one of the systems is arbitrarily fixed [9, Thm.4]. These results are summarized in the right column of Table 1.

5 Implications on conjunctive grammars

The complexity of equations over sets of numbers established above has direct implications on the complexity of conjunctive grammars over a one-letter alphabet.

Every conjunctive language can be parsed by a cubic-time algorithm and thus is in P [17], and some conjunctive languages over a multiple-letter alphabet are known to be P-complete [19]. The case of a unary alphabet is special, as it is known that no sparse language, in particular no unary language, can be P-complete unless DLOGSPACE = P [16,3], that is, unless the notion of P-completeness is trivial. However, from Theorem 2 one can infer the following result slightly weaker than P-completeness:

Corollary 1 *There exists an EXPTIME-complete set of numbers $S \subseteq \mathbb{N}$, such that the language $L = \{a^n \mid n \in S\}$ of unary notations of numbers from S is generated by a conjunctive grammar.*

Note that for every unary language generated by a conjunctive grammar, the corresponding set of numbers is in EXPTIME. The set constructed in Corollary 1 can thus be regarded as the computationally hardest among unary conjunctive languages.

This has a straightforward consequence referring to the complexity of parsing for conjunctive grammars. For context-free languages, it is known each of them is in NC^2, that is, can be parsed by a polynomial-size circuit of depth $O(\log^2 n)$, which was discovered independently by Brent and Goldschlager [1] and by Rytter [23]. The known examples of P-complete conjunctive languages imply that, unless P = NC, there are no polylogarithmic-time parallel parsing algorithm for conjunctive languages [19]. Now a similar result can be claimed with respect to grammars over a one-letter alphabet.

Corollary 2 *Unless* PSPACE = EXPTIME*, there is no logarithmic-space parsing algorithm for conjunctive languages over a unary alphabet.*

Indeed, having such an algorithm for the particular language L from Corollary 1 would give a polynomial-space algorithm for the EXPTIME-complete set S.

Let us now consider the complexity of the *compressed membership problem* for conjunctive grammars. This is a problem of testing whether a string w is generated by a grammar G, but unlike the ordinary membership problem, here the string w is given in a compressed form constructed by a data compression algorithm. The standard abstraction for data compression, which captures algorithms such as LZ78, LZW and the Huffman coding, is the notion of a *straight-line program* (SLP). Following Plandowski and Rytter [22], a straight-line program over the alphabet Σ is a context-free grammar $G_w = (\Sigma, N, P, S)$ with $L(G_w) = \{w\}$, and G_w is considered as a compressed representation of w. Note that the length of w may be exponentially larger than the description of the grammar G_w.

The compressed membership problem is defined as follows: "given a conjunctive grammar $G = (\Sigma, N, P, S)$ and a context-free grammar $G_w = (\Sigma, N, P, S)$ generating a singleton language $\{w\}$, determine whether $w \in L(G)$". The complexity of this

	Membership problem	Compressed membership problem
Deterministic finite automata	DLOGSPACE-complete	P-complete [14, 22]
Regular expressions	NLOGSPACE-complete	P-complete [14, 22]
Linear context-free grammars	NLOGSPACE-complete	PSPACE-complete [13, 22]
Context-free grammars	P-complete [10]	PSPACE-complete [13, 22]
Linear conjunctive grammars	P-complete [18, 19]	?
Conjunctive grammars	P-complete [18]	**EXPTIME-complete**
Context-sensitive grammars	PSPACE-complete	EXPSPACE-complete [13]

Table 2 Complexity of membership problems for grammars and automata.

problem for the common families of languages is known from the literature. For regular expressions, as well as for deterministic finite automata, the problem was shown to be in P by Plandowski and Rytter [22], while Markey and Schnoebelen [14] demonstrated its P-hardness already for a fixed regular language. Plandowski and Rytter [22] also showed that the compressed membership problem for context-free grammars is in PSPACE, and Lohrey [13] proved that it is PSPACE-hard even for a fixed deterministic linear context-free language. Furthermore, Lohrey [13] established the EXPSPACE-completeness of the same problem for context-sensitive grammars, showing that it is EXPSPACE-hard already for a fixed language.

Now the results of this paper can be used to establish the complexity of the same problem for conjunctive grammars.

Theorem 4 *The compressed membership problem for conjunctive grammars is EXPTIME-complete. It remains EXPTIME-complete for a fixed conjunctive language* $L_0 \subseteq a^*$.

Proof An exponential-time algorithm for this problem is straightforward. Given a conjunctive grammar G and a context-free grammar G_w with $L(G_w) = \{w\}$, the algorithm first decompresses the string w, that is, constructs it explicitly. Its length is at most exponential in the size of G_w. Then the known polynomial-time algorithm for solving the membership problem for a conjunctive grammar [18] is applied.

To show the EXPTIME-hardness of the problem for a particular language, let $S \subseteq \mathbb{N}$ be the set of numbers represented in Theorem 2. Define $L_0 = \{a^n \mid n \in S\}$, which is a conjunctive language by Corollary 1. Then the problem of testing whether a number n is in S can be reduced to the compressed membership problem in L_0 as follows.

Let $b_{\ell-1} \dots b_1 b_0$ with $b_i \in \{0,1\}$ be the binary notation of n, that is, $n = \sum_{i=0}^{\ell-1} b_i 2^i$. Let $i_1 < \ldots < i_k$ be all numbers with $b_{i_j} = 1$. Then the singleton language $\{a^n\}$ is generated by the following context-free grammar G_n:

$$\begin{aligned} S &\to A_{i_1} \dots A_{i_k} \\ A_0 &\to a \\ A_{i+1} &\to A_i A_i \quad (0 \leqslant i < i_k) \end{aligned}$$

Accordingly, the description of G_n is a yes-instance of the compressed membership problem for L_0 if and only if $n \in S$, which completes the reduction. □

This result is compared to the similar earlier cited results on other families of formal grammars in Table 2.

6 Conclusion

The first examples of non-periodic sets represented by equations over sets of numbers with union, intersection and addition have been discovered only recently [8,9], and now these equations were shown to be powerful enough to define EXPTIME-complete sets. At the same time, it remains an open question what is the exact family of sets of natural numbers defined by these equations. Besides the general knowledge that all representable sets are contained in DTIMESPACE($2^{n^2}, 2^n$), no methods of showing non-representability of particular sets are known. For instance, is it possible to define the set of all primes?

The related work on the complexity of expressions and circuits over sets of numbers [2,6,15,24,25,26] naturally suggests some further questions to study. In particular, by analogy to expressions and circuits over sets of integers (including negative numbers) studied by Travers [25], one can consider *equations* over sets of integers. No such equations have been studied before, and perhaps this research direction is worth being investigated.

References

1. R. P. Brent, L. M. Goldschlager, "A parallel algorithm for context-free parsing", *Australian Computer Science Communications*, 6:7 (1984), 7.1–7.10.
2. H.-G. Breunig, "The complexity of membership problems for circuits over sets of positive numbers", *Fundamentals of Computation Theory* (FCT 2007, Budapest, Hungary, August 27–30, 2007), LNCS 4639, 125–136.
3. J.-Y. Cai, D. Sivakumar, "Sparse hard sets for P: resolution of a conjecture of Hartmanis". *Journal of Computer and System Sciences*, 58:2 (1999), 280–296.
4. A. K. Chandra, D. C. Kozen, L. J. Stockmeyer, "Alternation", *Journal of the ACM*, 28:1 (1981), 114–133.
5. S. Ginsburg, H. G. Rice, "Two families of languages related to ALGOL", *Journal of the ACM*, 9 (1962), 350–371.
6. C. Glaßer, K. Herr, C. Reitwießner, S. D. Travers, M. Waldherr, "Equivalence problems for circuits over sets of natural numbers", *Computer Science in Russia* (CSR 2007, Ekaterinburg, Russia, September 3–7, 2007), LNCS 4649, 127–138.
7. D. T. Huynh, "Commutative grammars: the complexity of uniform word problems", *Information and Control*, 57:1 (1983), 21–39.
8. A. Jeż, "Conjunctive grammars can generate non-regular unary languages", *International Journal of Foundations of Computer Science*, 19:3 (2008), 597–615.
9. A. Jeż, A. Okhotin, "Conjunctive grammars over a unary alphabet: undecidability and unbounded growth", *Theory of Computing Systems*, to appear.
10. N. D. Jones, W. T. Laaser, "Complete problems for deterministic polynomial time", *Theoretical Computer Science*, 3:1 (1976), 105–117.
11. M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.
12. M. Kunc, "What do we know about language equations?", *Developments in Language Theory* (DLT 2007, Turku, Finland, July 3–6, 2007), LNCS 4588, 23–27.
13. M. Lohrey, "Word problems and membership problems on compressed words", *SIAM Journal on Computing*, 35:5 (2006), 1210–1240.
14. N. Markey, Ph. Schnoebelen, "A PTIME-complete matching problem for SLP-compressed words", *Information Processing Letters*, 90:1 (2004), 3–6.
15. P. McKenzie, K. Wagner, "The complexity of membership problems for circuits over sets of natural numbers", *Computational Complexity*, 16:3 (2007), 211–244.
16. M. Ogihara, "Sparse hard sets for P yield space-efficient algorithms", *Chicago Journal of Theoretical Computer Science*, 1996, article 2.
17. A. Okhotin, "Conjunctive grammars", *Journal of Automata, Languages and Combinatorics*, 6:4 (2001), 519–535.

18. A. Okhotin, "A recognition and parsing algorithm for arbitrary conjunctive grammars", *Theoretical Computer Science*, 302 (2003), 365–399.
19. A. Okhotin, "The hardest linear conjunctive language", *Information Processing Letters*, 86:5 (2003), 247–253.
20. A. Okhotin, "Decision problems for language equations", *Journal of Computer and System Sciences*, to appear.
21. A. Okhotin, "Unresolved systems of language equations: expressive power and decision problems", *Theoretical Computer Science*, 349:3 (2005), 283–308.
22. W. Plandowski, W. Rytter, "Complexity of language recognition problems for compressed words", in: J. Karhumäki, H. A. Maurer, G. Păun, G. Rozenberg (Eds.), *Jewels are Forever*, Springer, 1999, 262–272.
23. W. Rytter, "On the recognition of context-free languages", *Fundamentals of Computation Theory* (FCT 1985, Cottbus, Germany), LNCS 208, 315–322.
24. L. J. Stockmeyer, A. R. Meyer, "Word problems requiring exponential time", *5th Annual ACM Symposium on Theory of Computing* (STOC 1973, Austin, USA, April 30–May 2, 1973), 1–9.
25. S. D. Travers, "The complexity of membership problems for circuits over sets of integers", *Theoretical Computer Science*, 369:1–3 (2006), 211–229.
26. K. Yang, "Integer circuit evaluation is PSPACE-complete", *Journal of Computer and Systems Sciences*, 63:2 (2001), 288–303.