RAIRO-Inf. Theor. Appl. **41** (2007) 243–259
DOI: 10.1051/ita:2007019

DECIDABILITY OF CODE PROPERTIES

Henning Fernau[1], Klaus Reinhardt[2] and Ludwig Staiger[3]

Abstract. We explore the borderline between decidability and undecidability of the following question: "Let $\mathcal{C}$ be a class of codes. Given a machine $\mathfrak{M}$ of type X, is it decidable whether the language $L(\mathfrak{M})$ lies in $\mathcal{C}$ or not?" for codes in general, ω-codes, codes of finite and bounded deciphering delay, prefix, suffix and bi(pre)fix codes, and for finite automata equipped with different versions of push-down stores and counters.

Mathematics Subject Classification. 68Q45, 03D05, 94A45, 94B99.

1. Introduction

It is well-known that the property "$L = X^*$" is decidable for regular languages L over the alphabet X, whereas the same question is undecidable for context-free languages. The different character of regular and context-free languages also shows up for the question whether $L \subseteq X^*$ is a code, that is, freely generates the submonoid L^* of X^*. In Section 9 of the survey paper [13], it is stated that surprisingly few results of the following kind are known:

> Let $\mathcal{C}$ be a class of codes. Given a device D (automaton or grammar) of some fixed type, is it decidable whether the language $L(D)$ defined by D is in class $\mathcal{C}$ or not?

Keywords and phrases. Partially blind counter machines, prefix code, infix code, bifix code, deciphering delay, decidability.

[1] Fachbereich IV, Abteilung Informatik, Universität Trier, 54286 Trier, Germany; fernau@uni-trier.de
Most of the work on this paper has been undertaken while the author was with Wilhelm-Schickard-Institut für Informatik. The author is also still affiliated with The University of Newcastle, AUS.

[2] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, 72076 Tübingen, Germany; reinhard@informatik.uni-tuebingen.de

[3] Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, von-Seckendorff-Platz 1, 06099 Halle, Germany; staiger@informatik.uni-halle.de

The monograph [2] contains almost no information concerning this question. Known results are surveyed in Sections 3 and 9 of [13]; more details can be found in [12]. In these references, the undecidability results are mainly based on the undecidability of the Post correspondence problem (PCP). This yields undecidability results for devices like non-deterministic (one-turn) pushdown automata, leaving open this question for more restricted devices like one-counter or deterministic (one-turn) automata.

In our paper, we aim to show a distinctively sharp boundary between the automaton classes with a decidable or undecidable $\mathcal{C}$-code problem, respectively, for the following code classes $\mathcal{C}$: codes in general, ω-codes [25], codes of finite and bounded deciphering delay, prefix, suffix and bifix codes, because they form (aside from suffix codes) a natural decreasing chain of code classes. Moreover, according to Berstel and Perrin [2], p. 139, "the notion of deciphering delay appears at the very beginning of the theory of codes"[1]. Furthermore, these code classes seem to be important for applications like the computation of Hausdorff dimension of language-defined fractals as proposed in [6, 7, 26].

The sharp bounds in our paper are achieved by using instead of the PCP the undecidability of the halting problem for deterministic two-counter machines with empty input tape. This allows for showing the undecidability for classes of automata defining classes of languages of low complexity (see Fig. 1). Moreover, we strengthen our results by admitting side conditions, that is, asking for the undecidability of "$L(D) \in \mathcal{C}$" when it is known that $L(D) \in \mathcal{C}'$ for some class $\mathcal{C}'$ slightly larger than $\mathcal{C}$.

The paper is structured as follows: in the next section, we present the definitions necessary for the understanding of this paper. In Section 3, so-called C-chains are introduced as a basis for several proofs in Section 5. In Section 4, our decidability results are collected, while Section 5 contains the undecidability results. Finally, we summarize our results in Table 1.

2. Definitions

For basics in automata theory, we refer the reader to [1, 10, 11]. Especially, the notion of (deterministic) push-down automaton, (D)PDA for short, should be known, leading to the language classes (D)CF; if the (D)PDA is only allowed to make one turn of the push-down store during computation, we come to 1t(D)PDA, defining the language classes 1t(D)CF=(D)LIN. The regular languages are denoted by REG.

Furthermore, we obey the following conventions: $\mathbb{Z}$ is the set of integers; $\mathbb{N}$ is the set of natural numbers including zero; $sgn(x)$ is the sign of integer x, *i.e.*, $sgn(x) = 1$ if $x > 0$, $sgn(x) = -1$ if $x < 0$, and $sgn(0) = 0$; $\vec{0}$ is a multidimensional all-zero-vector. X^* is the free monoid over X, $e \in X^*$ denotes the empty word, $X^+ = X^* \setminus \{e\}$, $\sqsubseteq$ denotes the prefix relation in X^*.

[1]A good coverage of the corresponding literature can be found in [4].

2.1. Counter machines

Since definitions of counter automata are not standardized in the literature, we have to make the notions we use precise in this subsection, mostly following [9].

A *k-counter machine* $\mathfrak{M} = (Q, X, \delta, q_0, Q_f, k)$ consists of a finite set Q of *states*, a designated *initial* state q_0, a designated subset Q_f of *final* or *accepting* states, a finite *input* alphabet X and a finite *transition relation*

$$\delta \subseteq Q \times (X \cup \{e\}) \times \{0, 1, -1\}^k \times Q \times \{0, 1, -1\}^k.$$

Intuitively, this means that δ describes what $\mathfrak{M}$ should do when being in a specific state $q \in Q$, when reading the current input symbol $x \in X$ (or ignoring the input if $x = e$) and when finding that its ith counter c_i $(1 \leq i \leq k)$ is zero, strictly positive or strictly negative; this is matched by the value $sgn(c_i)$ in the ith place of the third component of the set product that describes δ. The last two components of that product describe the reaction of $\mathfrak{M}$ on the observed input; namely, $\mathfrak{M}$ would then go (in the next step) into a specified successor state and add $x_i \in \{0, 1, -1\}$ to counter c_i $(1 \leq i \leq k)$ as given by the ith place of the last component.

A *configuration* c of $\mathfrak{M}$ is a member of $Q \times X^* \times \mathbb{Z}^k$. The set of configurations is denoted by $C(\mathfrak{M})$. Especially, $c_0(w) = (q_0, w, \vec{0})$ is the *initial configuration for* w and $C_f = Q_f \times \{(e, \vec{0})\}$ is the set of *final configurations.*

Observe that we require here without loss of generality that all counters are zero at the end of a computation, a feature which will become essential for the special cases we consider in the following.

If $(q, a, u_1, \dots, u_k, q', x_1, \dots, x_k) \in \delta$ and $(q, aw, y_1, \dots, y_k)$ is a configuration of $\mathfrak{M}$ with $u_i = sgn(y_i)$ for $1 \leq i \leq k$, then we write

$$(q, aw, y_1, \dots, y_k) \vdash_{\mathfrak{M}} (q', w, y_1 + x_1, \dots, y_k + x_k).$$

If $a = e$, this is an *e-move.* $\vdash_{\mathfrak{M}}$ is a relation on $Q \times X^* \times \mathbb{Z}^k$. Its reflexive transitive closure is denoted by $\vdash_{\mathfrak{M}}^*$. The *language accepted by* $\mathfrak{M}$ is

$$L(\mathfrak{M}) = \{w \in X^* \,:\, \exists c_f \in C_f (c_0(w) \vdash_{\mathfrak{M}}^* c_f)\}.$$

We consider the following special cases of counter machines $\mathfrak{M} = (Q, X, \delta, q_0, Q_f, k)$:

- $\mathfrak{M}$ is *blind* if for each $q, q' \in Q$, $a \in X \cup \{e\}$, and for all u_i, v_i, x_i in $\{0, 1, -1\}$, it is true that

$$(q, a, u_1, \dots, u_k, q', x_1, \dots, x_k) \in \delta \iff (q, a, v_1, \dots, v_k, q', x_1, \dots, x_k) \in \delta\,. \quad (1)$$

  In other words, a blind counter machine is unable to check the signs of its counters during a computation. Only at the end, the acceptance condition checks whether all counters are zero.
- $\mathfrak{M}$ is *partially blind* if
  (1) $\delta \subseteq Q \times (X \cup \{e\}) \times \{0, 1\}^k \times Q \times \{0, 1, -1\}^k$ and

(2) for each $q, q' \in Q$, $a \in X \cup \{e\}$, and for all u_i, v_i in $\{0, 1\}$ and for all x_i in $\{0, 1, -1\}$, equation (1) is true.

So, a partially blind multi-counter machine may be viewed as a blind multi-counter machine which gets stuck should one of its counters decrease below zero.

- $\mathfrak{M}$ makes *one turn* if for any counter i, $1 \le i \le k$, and any subcomputation

$$(q_0, w, \vec{0}) \vdash^*_{\mathfrak{M}} (q_1, w_1, y_1, \ldots, y_k) \vdash^*_{\mathfrak{M}} (q_2, w_2, x_1, \ldots, x_k) \vdash^*_{\mathfrak{M}} (q_3, w_3, z_1, \ldots, z_k)$$

we find $x_i, y_i, z_i \ge 0$ and, if $y_i > x_i$, then $x_i \ge z_i$.
- $\mathfrak{M}$ is *deterministic* if for each $q \in Q$, $a \in X$ and for all u_i in $\{0, 1, -1\}$, it is true that

$$\begin{array}{rl} & |\{(q, a, u_1, \ldots, u_k, q', x_1, \ldots, x_k) \in \delta \mid q' \in Q, x_i \in \{0, 1, -1\}\}| \\ + & |\{(q, e, u_1, \ldots, u_k, q', x_1, \ldots, x_k) \in \delta \mid q' \in Q, x_i \in \{0, 1, -1\}\}| \\ \le & 1. \end{array}$$

In such a way, we are brought to the following language classes:

- the family [1t](D)BC of languages accepted by [one-turn] blind (deterministic) multi-counter machines;
- the family [1t](D)PBC of languages accepted by [one-turn] partially blind (deterministic) multi-counter machines;
- we could, in addition, specify the number of counters in our notations by setting a numeral in front of C, *e.g.*, D1C is the family of deterministic one-counter languages[2].

We briefly recall three non-trivial facts on counter machines:

(1) From the decidability of the reachability problem for Petri nets [14, 19], the decidability of the emptiness problem for (partially) blind counter machines results follows, see Theorem 6 in [9][3].

(2) According to Minsky [20, 21], *cf.* also Section 7.8 in [11], the halting problem for two-counter machines is undecidable, even if one takes D2C machines with only one accepting state whose counters never get below zero and to which is given the empty word as input. Such machines have a unique final configuration c_f, *i.e.*, $C_f(e) = \{c_f\}$.

Furthermore, we may assume w.l.o.g. that the machine never enters the start state q_0 and never leaves the final state q_f again. We will call such a machine a D2CA *in normal form.*

(3) According to Greibach [9], Theorem 2, the family BC of languages accepted by blind multi-counter machines coincides with the family 1tC of languages accepted by one-turn multi-counter machines. The proof does not transfer,

[2]PB1C is called "restricted one-counter languages" in [1], and 1C is called "iterated counter languages" in [10]. One-turn counter machines are called "reversal-bounded" in [9].

[3]The emptiness problem for PBLIND(n) is equivalent to the emptiness problem for PBLIND=PBC by adding an additional blank symbol replacing e-moves.

$$\begin{tikzcd}
 & & & \mathrm{D2C} = \mathrm{RE} & \\
 & & & \mathrm{CF} \arrow[u] & \\
\mathrm{PBC} \arrow[uurrr] & & \mathrm{1C} \arrow[ur] & & \mathrm{LIN} \arrow[ul] \\
 & & & \mathrm{DCF} \arrow[uu] & \\
\mathrm{BC} = \mathrm{1tC} \arrow[uu, dotted] & & \mathrm{D1C} \arrow[uu] \arrow[ur] & & \mathrm{DLIN} \arrow[ul] \arrow[uu] \\
 & & & \mathrm{1tD1C} \arrow[ulll] \arrow[ul] \arrow[ur] & \\
 & & & \mathrm{1tDB1C} \arrow[u, dotted] & \\
 & & & \mathrm{REG} \arrow[u] &
\end{tikzcd}$$

FIGURE 1. Classes of languages of low complexity.

neither to the deterministic case nor to the case of a fixed number of counters. In fact,

$$L := \{ w \in \{a,b\}^* : |w|_a = |w|_b \} \in \mathrm{DB1C},$$

where $|w|_x$ gives the number of occurrences of letter x in string w. L cannot be accepted by one-counter machines which only make a finite number of turns.

(4) Blind counter machines can be simulated by partially blind counter machines but not vice versa, see Theorems 3 and 4 in [9].

The second fact will be the main tool for showing our undecidability results. In our constructions, we will use the *quasi unary encoding* $\gamma(c) := q|^z\#$ with $z = 2^n \cdot 3^m$ for a configuration $c = (q, n, m)$ of a 2C machine with empty input[4]. "Quasi unary" means that except for the first letter $q \in Q$ and the endmarker $\#$ the code word $\gamma(c)$ is unary. Since there is only a finite number of states $q \in Q$, a configuration c can be read, stored and compared to some previously stored encoded configuration by a counter automaton as well as by a one-turn push-down automaton. γ can be easily interpreted as a homomorphism mapping sequences of configurations to words over the finite alphabet $Q \cup \{|, \#\}$.

We conclude this section with presenting a diagram of inclusion relations between the classes of languages mentioned above and its relations to regular, linear, context-free and recursively enumerable languages. In the diagram, solid lines indicate known strict inclusions, while dotted lines indicate inclusions not known to be strict.

[4]The notation $|^z$ refers to the symbol $|$ repeated z times.

2.2. Codes

A language $C \subseteq X^*$ is called *code* over X if for all $n, m \in \mathbb{N}$ and $x_1, \dots, x_n$; $y_1, \dots, y_m \in C$ the condition $x_1 \dots x_n = y_1 \dots y_m$ implies $n = m$ and $x_i = y_i$ for all $1 \leq i \leq n$.

A code C is called *prefix code* if for all $x, y \in C$, $x \sqsubseteq y$ implies $x = y$, *i.e.*, the prefix relation restricted to $C \subseteq X^+$ is the identity. Observe that in fact any language $C \subseteq X^+$ satisfying that the prefix relation restricted to C is the identity is a (prefix) code.

A code C is called *suffix code* if the suffix relation restricted to C is the identity.

A language C is called *bifix code* iff C is both a prefix and a suffix code[5].

Notice that any subset of a code (prefix code, suffix code, bifix code, resp.) is again a code (prefix code, suffix code, bifix code, resp.).

Example 2.1. The language $C_Q := \{q|^z\# : q \in Q \wedge z \in \mathbb{N}\}$ containing the quasi unary encodings (as well as non-encoding words if z has factors other than 2 or 3) from above is a bifix code. Hence, the language of all quasi unary encodings itself is a bifix code.

According to [2, 5, 25], a code C has a *deciphering delay* $m \geq 0$ (m-d.d. for short) iff for all $w, w' \in C$ the relation $w \cdot v_1 \cdots v_m \sqsubseteq w' \cdot u$ where $v_1, \dots, v_m \in C$ and $u \in C^*$ implies $w = w'$. We say that a code C has *bounded deciphering delay* (b.d.d. for short) provided C has m-d.d. for some $m \in \mathbb{N}$. Observe that C is a prefix code iff C has 0-d.d.

The following obvious lower bound to the deciphering delay is sometimes useful.

Property 2.2 (lower bound)**.** *If $C \subseteq X^+$ is a code and if there are words w, w', $v_1, \dots, v_i \in C$ such that $w \neq w'$ and $w \cdot v_1 \cdots v_i \sqsubseteq w' \cdot u$ for some $u \in C^*$ then C has a deciphering delay of at least $i + 1$.*

A code C is said to have *finite deciphering delay* (f.d.d. for short), provided for every $w \in C$ there is an $m_w \in \mathbb{N}$ such that, for every $w' \in C$, the relation $w \cdot v_1 \cdots v_{m_w} \sqsubseteq w' \cdot u$ where $v_1, \dots, v_{m_w} \in C$ and $u \in C^*$ implies $w = w'$. By $m_C(w)$ we denote the smallest value m_w, possible for $w \in C$.

A code C is called an *ω-code*, provided $\prod_{i=1}^{\infty} w_i = \prod_{i=1}^{\infty} v_i$ where $w_i, v_i \in C$, $i = 1, 2, \dots$ implies $w_i = v_i$ for all $i = 1, 2, \dots$

It is known that every code of bounded deciphering delay is a code of finite deciphering delay, and a code of finite deciphering delay is an ω-code, whereas the converse is not true in both cases (*cf.* [25]).

For codes of bounded deciphering delay we can prove the following.

Lemma 2.3. *A language $C \subseteq X^+$ is a code of deciphering delay m if and only if for all $w, w', v_1, \dots, v_m, v_1', \dots, v_j' \in C$ where $j \leq m$ the relation $w \cdot v_1 \cdots v_m \sqsubseteq w' \cdot v_1' \cdots v_j'$ implies $w = w'$.*

Proof. From the definition above, the necessity of our condition is immediate.

[5]Such codes are named biprefix codes in [2].

Assume now that C satisfies the condition of the lemma and that $w \cdot v_1 \cdots v_m \sqsubseteq w' \cdot u$ where $w, w', v_1, \ldots, v_m \in C$ and $u \in C^*$. Let $u = v'_1 \cdots v'_j$ where j is assumed to be minimal, that is, $w \cdot v_1 \cdots v_m \not\sqsubseteq w' \cdot v'_1 \cdots v'_{j-1}$ if $j \geq 1$.

If $j \leq m$ then the condition implies $w = w'$.

If $j > m$ then by the minimality of j we conclude from $w \cdot v_1 \cdots v_m \sqsubseteq w' \cdot v'_1 \cdots v'_j$ that $w' \cdot v'_1 \cdots v'_m \sqsubseteq w' \cdot v'_1 \cdots v'_{j-1} \sqsubset w \cdot v_1 \cdots v_m$, whence again $w' = w$. □

3. C-Chains

In this section, following an idea of Levenshtejn (*cf.* [16] and also Sect. 2.2.1 of [17] or [25]), we introduce *Levenshtejn's relation* $\prec$ on X^* which useful in the study of codes. It describes the possibilities of double factorizations of code messages. For a subset $C \subseteq X^+$, define $\prec$ as follows:

$$\begin{aligned} w \prec_1 v \quad &:\Leftrightarrow \quad wv \in C \ , \\ w \prec_2 v \quad &:\Leftrightarrow \quad w \in C \cdot v \quad \text{and} \quad \prec := \prec_1 \cup \prec_2 \ . \end{aligned}$$

We consider C-chains, that is, sequences of the form

$$\Gamma := u_1 \prec_1 u_2 \prec_{k_2} u_3 \prec \ldots \prec_{k_{n-1}} u_n \ , \tag{2}$$

where $u_1 \in C,\ u_2 \neq e$ and $k_i \in \{1, 2\}$ for $i \geq 2$[6].

We call a C-chain Γ *nontrivial* provided $n \geq 2$. Observe that for a nontrivial C-chain $\Gamma = u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_n$ we have $u_1 \sqsubset u_1 \cdot u_2$ and $u_1, u_1 \cdot u_2 \in C$.

The following theorem shows a close connection between C-chains and double factorizations.

Theorem 3.1. *Let $C \subseteq X^+$.*

(1) *If there are families $(v_k)_{k=1}^i$ and $(w_k)_{k=1}^j$ of words $v_k, w_k \in C$ with $w_1 \neq v_1$ and $w_1 \cdots w_{j-1} \sqsubseteq v_1 \cdots v_i \sqsubseteq w_1 \cdots w_j$ then there is a C-chain $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_{i+j}$ such that $\{u_1, u_1 \cdot u_2\} = \{v_1, w_1\}$, $v_1 \cdots v_i \cdot u_{i+j} = w_1 \cdots w_j$ and $|u_{i+j}| \leq |w_j|$.*

(2) *If a sequence $(u_k)_{k=1}^n$, $n \geq 2$ is a C-chain $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_n$, then there are words $w_1, \ldots, w_j, v_1, \ldots, v_i \in C$ where $w_1 \neq v_1$, $\{u_1, u_1 \cdot u_2\} = \{w_1, v_1\}$, $|u_n| \leq |w_j|$ and $i + j = n$ such that*

$$v_1 \cdots v_i \cdot u_n = w_1 \cdots w_j \ \text{and}\ |u_n| \leq |w_j| \ . \tag{3}$$

Proof. The proof is by induction on $n = i + j$. In both cases the assertion is obvious if $n = i + j \leq 2$.

1. Let $w_1 \cdots w_{j-1} \sqsubseteq v_1 \cdots v_i \sqsubseteq w_1 \cdots w_j$. We distinguish two cases.

If $w_1 \cdots w_{j-1} \sqsubseteq v_1 \cdots v_{i-1} \sqsubseteq w_1 \cdots w_j$ then, by the induction hypothesis, there is a C-chain $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_{i+j-1}$ such that $\{u_1, u_1 \cdot u_2\} = \{v_1, w_1\}$ and $v_1 \cdots v_{i-1} \cdot u_{i+j-1} = w_1 \cdots w_j$. Define u_{i+j} via the equation $u_{i+j-1} = v_i \cdot u_{i+j}$.

[6]Sometimes we shall append the values $k_i \in \{1, 2\}$ for easier orientation.

Then $u_{i+j-1} \prec_2 u_{i+j}$, and we can continue our C-chain to length $i+j$ satisfying the required properties.

In case $v_1 \cdots v_{i-1} \sqsubseteq w_1 \cdots w_{j-1} \sqsubseteq v_1 \cdots v_i$ using the induction hypothesis we obtain a C-chain $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_{i+j-1}$ such that $w_1 \cdots w_{j-1} \cdot u_{i+j-1} = v_1 \cdots v_i$ and $|u_{i+j-1}| \leq |v_i|$. Consequently, $u_{i+j-1} \sqsubseteq w_j$, and defining u_{i+j} via the equation $u_{i+j-1} \cdot u_{i+j} = w_j$ we obtain the required prolongation $u_{i+j-1} \prec_1 u_{i+j}$ of our C-chain $u_1 \prec_1 u_2 \prec \ldots \prec u_{i+j-1}$.

2. Assume now that the assertion holds for arbitrary C-chains of length n and let $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_n \prec u_{n+1}$ be a C-chain of length $n+1$. According to our assumption on the initial part $u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_n$ we can find words $v_1, \ldots, v_i, w_1, \ldots, w_j \in C$ such that $v_1 \cdots v_i \cdot u_n = w_1 \cdots w_j$, $w_1 \neq v_1$,$\{u_1, u_1 \cdot u_2\} = \{w_1, v_1\}$, $|u_n| \leq |w_j|$ and $i + j = n$. Again we consider two cases.

If $u_n \prec_1 u_{n+1}$, that is, $u_n \cdot u_{n+1} \in C$ we set $v_{i+1} := u_n \cdot u_{n+1}$ and obtain $w_1 \cdots w_j \cdot u_{n+1} = v_1 \cdots v_{i+1}$.

If $u_n \prec_2 u_{n+1}$ there is a word $v_{i+1} \in C$ such that $u_n = v_{i+1} \cdot u_{n+1}$, and we obtain $v_1 \cdots v_{i+1} \cdot u_{n+1} = w_1 \cdots w_j$. □

Theorem 3.1 makes the following equivalences obvious, *cf.* also [2,25].

Property 3.2 (The Sardinas-Patterson Theorem)**.** *C is a code iff there is no nontrivial C-chain terminating with a word $u_n \in C$.*

If a C-chain $\Gamma = u_1 \prec_1 u_2 \prec u_3 \prec \ldots \prec u_i$ terminates with $u_i \in C$, then we may proceed by adding $u_i \prec_1 u_{i+1} = e \prec_1 u_i \prec_1 u_{i+1} = e \prec \ldots$. In general, a C-chain contains the empty word $u_i = e$ if and only if its preceding entry u_{i-1} is in C. In the sequel, we will refer to C-chains not containing the empty word as *proper*.

We call a proper C-chain Γ *maximal* provided Γ cannot be extended to a proper C-chain.

In the same way as in equation (2) we can define infinite C-chains. Thus, a proper infinite C-chain is always maximal.

Property 3.3. *C is an ω-code iff there is no infinite C-chain.*

Let $\ell_C(w)$ denote supremum over the lengths of all C-chains starting with $w \prec_1 u_2$ or with $u_1 \prec_1 u_2$ where $w = u_1 \cdot u_2$.

If $m_C(w) > i$ then there are $w, w', v_1, \ldots, v_i \in C$, $w \neq w'$ and a word $u \in C^*$ such that $w \cdot v_1 \cdots v_i \sqsubseteq w' \cdot u$. According to Theorem 3.1 we have a C-chain starting with $w \prec_1 u_2$ or with $w' \prec_1 u_2$ where $w = w' \cdot u_2$ of length $\geq i+2$. Thus, $\ell_C(w) \geq i+2$, and we obtain the following.

Property 3.4. *If $C \subseteq X^*$ is a code then $\forall w(w \in C \rightarrow \ell_C(w) \geq m_C(w) + 1)$.*

This yields a connection to codes having finite deciphering delay.

Property 3.5. *Let $\ell_C(w) < \infty$ for every $w \in C$. Then C has f.d.d.*

As we shall see in the proof of Theorem 5.11, the converse is not true.

If C has m-d.d.,then, in view of Lemma 2.3, we can derive a tighter relationship.

Property 3.6. *If C has deciphering delay m then $\ell_C(w) \leq 2m+1$ for every $w \in C$, and if $\ell_C(w) \leq n$ for every $w \in C$ then C has deciphering delay $n-1$.*

Proof. From Property 2.2 we know that in equation (3) $i \le m$ and $j \le m+1$, whence $n = i + j \le 2m + 1$.

The bound $m \le \sup\{\ell_C(w) - 1 : w \in C\}$ is derived above in Property 3.4. □

4. Decidability results

In [10], p. 355, Problem 7, it was mentioned that for DCF the property to be a prefix code is decidable, whilst the prefix code property is known to be undecidable for CF, see [10], p. 262, Problem 6. Although this result seems to be folklore, we give an outline of the proof in the following.

Theorem 4.1. *It is decidable for a DCF $L \subseteq X^+$ whether L is a prefix code.*

Proof. Let $\mathfrak{M}$ be a DPDA accepting L. Modify $\mathfrak{M}$ as follows in order to obtain a DPDA $\mathfrak{M}'$ such that $L(\mathfrak{M}') = \emptyset$ iff L is a prefix code: $\mathfrak{M}'$ has two copies of the set of states of $\mathfrak{M}$; it starts working like $\mathfrak{M}$ on the first copy, switching deterministically to the second copy when first reaching a final state of $\mathfrak{M}$ upon actually reading the next (non-empty) symbol; then, it proceeds to work on the second copy like $\mathfrak{M}$; the accepting states of $\mathfrak{M}$ in the second copy are the accepting states of $\mathfrak{M}'$. This makes sure that a word x is accepted by $\mathfrak{M}'$ if and only if x has a proper prefix that is accepted by $\mathfrak{M}$. □

Obviously, the previous construction is applicable to all deterministic machine models with a decidable emptiness problem, like deterministic stack automata [8], deterministic set automata [15], etc[7].

Theorem 5.8 below shows that we cannot expect to sharpen the previous theorem even from 0-d.d. decidability to 1-d.d. decidability. For (partially) blind counters, however, we obtain an even stronger result. This result is interesting also in the following respect.
Although, in view of Theorems 5.1 and 5.9, we cannot decide whether a language $L \subseteq X^+$ accepted by a partially blind counter automaton is a code or a code of bounded deciphering delay, we can decide whether it has a given deciphering delay.

Theorem 4.2. *For every fixed $m \ge 0$, it is decidable for a PBC $L \subseteq X^+$ whether L is a code of deciphering delay m or not.*

Proof. Let $\mathfrak{M} = (Q, X, \delta, q_0, \{q_f\}, k)$ be a PBCA. (Since $\mathfrak{M}$ is nondeterministic, we may assume that it has only one accepting state q_f.) First we build an automaton $\mathfrak{A}$ as the marked union of $m+1$ copies of $\mathfrak{M}$ using all disjoint counters and connecting them by adding e-moves $(q_{f,i}, e, \vec{0}, q_{0,i+1})$ from the ith copy of q_f to the $i+1$st copy of q_0 for all $0 \le i \le m$. Additionally we add a new final state $\hat{q}_f$ and the transitions $(q_{f,m}, e, \vec{0}, \hat{q}_f)$ and $(\hat{q}_f, a, \vec{0}, \hat{q}_f)$. If the finite control is within a state of the ith copy of Q, then it may only increment or decrement counters from

[7] Conversely, if for a class of languages $\mathcal{L}$ closed under union with finite languages and concatenation from the left with finite languages, the property of being a prefix code for $L \in \mathcal{L}$ is decidable, then also the emptiness problem for $\mathcal{L}$ is decidable, see Section 6.

the ith copy of the k counters of $\mathfrak{M}$. Thus $\mathfrak{A}$ has $k(m+1)$ counters, starts in $q_{0,0}$ and reaches $q_{f,m}$ iff it reads a word in L^{m+1} and finally $\hat{q}_f$ iff it reads a word in $L^{m+1}X^*$.

Consider the canonical product automaton $\mathfrak{M}' = \mathfrak{A} \times \mathfrak{A}$ with $2k(m+1)$ counters. $\mathfrak{M}''$ is obtained by enclosing an additional finite control, which ensures that the use of the transition containing $(q_{f,0}, e, \vec{0}, q_{0,1})$ in the first component and the transition containing $(q_{f,0}, e, \vec{0}, q_{0,1})$ in the second component is separated by a non-e-transition. Now, $L(\mathfrak{M}'')$ is empty iff for all $w, w' \in L$, $w \neq w'$, $wL^mX^* \cap w'L^mX^*$ is empty iff L has m-d.d, *cf.* Lemma 2.3. □

The previous construction should work for all automata classes $\mathcal{A}$ with a decidable emptiness problem, if $\mathcal{A}$ is closed under the product automaton construction.

PBC is closed under mirror image, so that we immediately obtain:

Corollary 4.3. *It is decidable for a PBC $L \subseteq X^+$ whether L is a prefix code or whether L is a suffix code or whether L is a bifix code.*

In fact, the last argument should work for all classes $\mathcal{A}$ of nondeterministic automata (like PBC) with "reasonable" storage types: such classes are closed under mirror image, since nondeterminism allows to "trace back" the computation of a machine $\mathcal{M}$ on a word $w = a_1 \dots a_n$ by another machine $\mathcal{M}'$ on the mirror word $w^R = a_n \dots a_1$.

By way of contrast, observe that often languages classes defined via deterministic machine models are not closed under mirror image, so that we cannot conclude that say for DCF languages, the suffix code property is decidable. In fact, this property is undecidable for DCF (and even more restricted deterministic language classes), as we will see in the next section.

We remark that the preceding three results are also valid in case of languages L containing the empty word. For such languages, the answer has to be "no", since no language containing the empty word is a code. Since $e \in L(A)$? can be tested algorithmically for all automata classes considered in this section, we can cope with arbitrary languages $L \subseteq X^*$, too.

Now, we turn to several undecidability results.

5. Undecidability results

5.1. Blind counters

The proofs in this subsection rely on the properties listed in Section 3.

Theorem 5.1. *Let $L \in$ 1tDB1C. Then, the property "L is a code" is undecidable.*

Proof. Let $\mathfrak{M} = (Q, \delta, q_0, \{q_f\})$ with $q_0 \neq q_f$ be a D2CA with empty input in normal form. We use the quasi unary encoding for the configurations of $\mathfrak{M}$ and define our language $L \subseteq \Big(Q \cup \{\#\,,|\}\Big)^*$ as follows:

$$\begin{aligned}
L &:= L_0 \cup L_1 \cup L_f, \quad \text{where} \\
L_0 &:= \{q_0|\#\} = \{\gamma(c_0)\}, \\
L_1 &:= \{\gamma(c)\gamma(c') \,:\, c, c' \in C(\mathfrak{M}) \wedge c \vdash_{\mathfrak{M}} c'\}, \\
L_f &:= \{q_f|\#\} = \{\gamma(c_f)\}.
\end{aligned}$$

It is readily seen that $L_1 \cup L_f$ is a prefix code and that $L \in$ 1tDB1C[8].

Since the words in L_1 link the configurations of $\mathfrak{M}$ to their successor configurations, and since $L_1 \cup L_f$ contains exactly one word with prefix $q_0|\#$, it is immediate that L admits exactly one nontrivial maximal L-chain Γ:

$$\Gamma = q_0|\# \prec_1 q_1 w_1\# \prec_1 q_2 w_2 \# \prec_1 \dots\,,$$

where $(q_i, n_i, m_i) \vdash_{\mathfrak{M}} (q_{i+1}, n_{i+1}, m_{i+1})$ when w_j is the word consisting of $2^{n_j} \cdot 3^{m_j}$ letters $|$. Thus L is a code iff this L-chain does not end with $q_f|\#$, that is, the computation of $\mathfrak{M}$ does not halt. □

The previous result strengthens the undecidability of the code property shown for linear languages in [13].

With a slight modification of the construction of Theorem 5.1 we obtain the following.

Theorem 5.2. *For languages $L \in$ 1tDB1C, it is undecidable whether L is an ω-code, even if we suppose L to be a code.*

Proof. The language $C := L_0 \cup L_1$, where L_0 and L_1 are defined in the proof of Theorem 5.1, is a code with exactly one infinite C-chain iff the machine $\mathfrak{M}$ does not halt. Otherwise, there is only one finite maximal C-chain of length greater than 1. □

Corollary 5.3. *For languages $L \in$ 1tDB1C, it is undecidable whether L is a code of finite (bounded) deciphering delay, even if we suppose L to be a code.*

Proof. The code C constructed in the proof of Theorem 5.2 is an ω-code iff C has finite deciphering delay iff C has bounded deciphering delay. □

5.2. Other cases

In the case of nondeterministic one-counter and linear languages we obtain a series of results concerning non-decidability of questions related to the deciphering delay of codes, thereby sharpening Theorem 9.5 of [13] in parts.

[8]Details work like in the proof of Lemma 5.1 in [22]. The counter is used to check a multiplication by 2 or 3 simulating an increment by reading || or ||| in the second configuration for every letter | in the first configuration and vice versa for a division simulating a decrement. Divisibility by 2 or 3 simulating a zero-test can be checked already by the finite control.

We start with a general construction. Let, as above, $\mathfrak{M} = (Q, \delta, q_0, \{q_f\})$ be a deterministic two-counter machine with empty input. We can assume $q_0 \neq q_f$ and $\mathfrak{M}$ to be in normal form. Recall that the encoding γ can be viewed as a morphism. With the help of the regular language

$$C_0 := \gamma(c_0c_0)\{\, \gamma(c) \,:\, c \in C(\mathfrak{M}) \setminus \{c_0, c_f\} \,\}^*\gamma(c_fc_f),$$

we define the following deterministic one-counter and linear languages derived from the quasi-unary encoding of the configurations of $\mathfrak{M}$ (observe that C_0 is a bifix code):

$$\begin{array}{lll} C_1(\mathfrak{M}) & := & \gamma(c_0)\{\gamma(cc') : c, c' \in C(\mathfrak{M}) \wedge c \vdash_{\mathfrak{M}} c'\}^*\gamma(c_f) \cap C_0 \\ C_2(\mathfrak{M}) & := & \gamma(c_0c_0)\{\gamma(cc) : c \in C(\mathfrak{M})\}^*\gamma(c_fc_f) \cap C_0 \\ L_1(\mathfrak{M}) & := & \{\gamma(c_nc_{n-1}\dots c_2c_1)\$\gamma(c_0c'_1c'_2\dots c'_{n-1}c'_n) \,: \\ & & \quad n \in \mathbb{N} \wedge c_i, c'_i \in C(\mathfrak{M}) \wedge c_i \vdash_{\mathfrak{M}} c'_i \text{ for } 1 \leq i \leq n\} \\ L_2(\mathfrak{M}) & := & \{\gamma(c_nc_{n-1}\dots c_2c_1)\$\gamma(c_1c_2\dots c_{n-1}c_n)\gamma(c_f) \,: \\ & & \quad n \in \mathbb{N} \wedge c_i \in C(\mathfrak{M}) \text{ for } 1 \leq i \leq n\}. \end{array}$$

One easily observes that $C_1(\mathfrak{M})$ and $C_2(\mathfrak{M})$ are deterministic one-counter languages, whereas $L_1(\mathfrak{M})$ and $L_2(\mathfrak{M})$ are deterministic linear languages.

Since $\mathfrak{M}$ is a deterministic two-counter machine with empty input in normal form, the following lemma is valid.

Lemma 5.4. *The languages $C_1(\mathfrak{M}) \cup C_2(\mathfrak{M})$ and $L_1(\mathfrak{M}) \cup L_2(\mathfrak{M})$ are bifix codes.*

The following lemma is crucial for our non-decidability results.

Lemma 5.5. *The deterministic two-counter machine with empty input in normal form $\mathfrak{M}$ halts iff $C_1(\mathfrak{M}) \cap C_2(\mathfrak{M}) \neq \emptyset$ iff $L_1(\mathfrak{M}) \cap L_2(\mathfrak{M}) \neq \emptyset$.*

Proof. We prove only that $\mathfrak{M}$ halts iff $L_1(\mathfrak{M}) \cap L_2(\mathfrak{M}) \neq \emptyset$, the proof of the other equivalence being similar.

We have $w = \gamma(c_nc_{n-1}\dots c_2c_1)\$\gamma(c_0c'_1c'_2\dots c'_{n-1}c'_n) \in L_1(\mathfrak{M}) \cap L_2(\mathfrak{M})$ if and only if first $c_1 = c_0$, $c_{i+1} = c'_i$ for $1 \leq i < n$ and $c'_n = c_f$, because $w \in L_2(\mathfrak{M})$, and then $c_0 \vdash_{\mathfrak{M}} c_1 \vdash_{\mathfrak{M}} \cdots \vdash_{\mathfrak{M}} c_n \vdash_{\mathfrak{M}} c_f$, because $w \in L_1(\mathfrak{M})$. This observation makes the assertion obvious. □

The developed apparatus enables us to prove the results.

Theorem 5.6. *It is undecidable for $L \in 1C$ (or $L \in LIN$) whether L is a prefix code, even if we know that L is a suffix code and has deciphering delay 1.*

Proof. Again, we simulate a D2C machine $\mathfrak{M}$ with empty input in normal form using quasi unary encoding. Let

$$L := C_1(\mathfrak{M}) \cup C_2(\mathfrak{M})\,\#\ ,$$

where $C_1(\mathfrak{M}), C_2(\mathfrak{M})$ are defined above. Obviously, $L \subseteq C_0 \cup C_0\,\#$ is a suffix code.

By Lemma 5.4 $C_1(\mathfrak{M}) \cup C_2(\mathfrak{M})$ is a prefix code, hence a word $w_1 \in L$ can only be a prefix of a word $w_2 \in L$ iff $w_1 \in C_1(\mathfrak{M})$, $w_2 \in C_2(\mathfrak{M})\,\#$ and $w_2 = w_1\,\#$, that is, iff $C_1(\mathfrak{M}) \cap C_2(\mathfrak{M}) \neq \emptyset$. Thus L is a prefix code iff $\mathfrak{M}$ halts, and the latter is undecidable.

The relation $w_2 = w_1\,\#$ results in an L-chain $w_1 \prec_1 \#$ which has no continuation. Thus, in view of Property 3.6, L has 1-d.d. (and is a suffix code).

The proof for LIN proceeds analogously using $L := L_1(\mathfrak{M}) \cup L_2(\mathfrak{M})\,\#$. □

For suffix codes, we have a stronger result using a simple modification of the previous construction.

Theorem 5.7. *It is undecidable for a $L \in$ D1C (or $L \in$ DLIN) which is a prefix code whether L is a suffix (and hence bifix) code.*

Proof. Let $\mathfrak{M}$ be a D2CA with empty input in normal form. The language

$$L := C_1(\mathfrak{M}) \cup \$C_2(\mathfrak{M})$$

is a prefix code and in D1C. Since according to Lemma 5.4 $C_1(\mathfrak{M}) \cup C_2(\mathfrak{M})$ is a suffix code, a word $w_1 \in L$ is a suffix of $w_2 \in L$ iff $w_1 \in C_1(\mathfrak{M})$, $w_2 \in \$C_2(\mathfrak{M})$ and $w_2 = \$w_1$, that is $w_1 \in C_1(\mathfrak{M}) \cap C_2(\mathfrak{M})$.

Similarly, the linear case can be treated. □

The undecidability results for linear languages proved in the preceding two theorems are already shown in [13], Section 9, using constructions based on Post's correspondence problem.

Again, simple modifications yield the next theorem:

Theorem 5.8. *For $L \in$ D1C (or $L \in$ DLIN), it is undecidable whether L is a code of deciphering delay $m \geq 1$, even if we suppose that L has $m+1$-d.d.*

Proof. Let $\mathfrak{M}$ be a D2CA with empty input in normal form and consider

$$L := C_1(\mathfrak{M}) \cup \$^m C_2(\mathfrak{M})\,\# \cup \{\$\}\ .$$

It is obvious that we have only the following L-chains of length $m+1$:

$$\$ \prec_1 \$^{m-1} w\,\# \prec_2 \$^{m-2} w\,\# \prec_2 \cdots \prec_2 w\,\# \quad \text{where } w\,\# \in C_2(\mathfrak{M})\,\# \qquad (4)$$

corresponding to $\$ \cdot \$^{m-1} \sqsubseteq \$^m w\,\#$ for $w \in C_2(\mathfrak{M})$. Thus Property 2.2 shows that L has at least deciphering delay m.

Moreover, L-chains of length $m+2$, $\$ \prec_1 \$^{m-1} w\,\# \prec_2 \$^{m-2} w\,\# \prec_2 \cdots \prec_2 v \prec_2 w\,\#$, exist iff $v \sqsubseteq w\,\#$ for some $v \in C_1(\mathfrak{M})$. Other L-chains longer than the ones in equation (4) do not exist. In view of Lemma 5.4, the condition $v \sqsubseteq w\,\#$ for some $v \in C_1(\mathfrak{M})$ is equivalent to $v \in C_1(\mathfrak{M}) \cap C_2(\mathfrak{M})$. Thus L has $(m+1)$-d.d. but not m-d.d. iff $C_1(\mathfrak{M}) \cap C_2(\mathfrak{M}) \neq \emptyset$ iff $\mathfrak{M}$ halts.

In order to meet the 1-turn restriction, for $L \in \mathrm{DLIN}$ we use again the languages $L_1(\mathfrak{M})$ and $L_2(\mathfrak{M})$:

$$L := L_1(\mathfrak{M}) \cup \$^m L_2(\mathfrak{M})\,\# \cup \{\$\}\ ,$$

and the proof proceeds in the same way as for the one-counter case. □

Now we turn to the undecidability of the bounded delay property.

Theorem 5.9. *For languages $L \in$ 1tDB1C it is undecidable whether L has b.d.d., even if we suppose L to have f.d.d.*

Proof. In order to guarantee that L has finite deciphering delay, we change the construction in the proof of Theorem 5.2 as follows: we encode the configuration (q, n, m) of $\mathfrak{M}$ aiming to allow at most t further steps as $\gamma_t^u(q, n, m) := q|^z\#$ with $z = 2^n \cdot 3^m \cdot 5^t \cdot u$ and define $C \subseteq \Big(Q \cup \{\#, |\}\Big)^*$ in the following way:

$$\begin{aligned} C &:= L_0 \cup L_1, \quad \text{where} \\ L_0 &:= \{q_0|^{5^t u}\# : t \in \mathbb{N} \text{ and } u \text{ is not divisible by 2, 3 or 5}\}, \\ L_1 &:= \{\gamma_t^u(q, n, m)\gamma_{t-1}^u(q', n', m') : q' \neq q_f \wedge (q, n, m) \vdash_{\mathfrak{M}} (q', n', m') \wedge t \geq 1\}. \end{aligned}$$

The deciphering delay of a word in C is determined by the t in the first encoding, so that C has f.d.d. If and only if $\mathfrak{M}$ halts after at most s steps when having started from an arbitrary configuration, the deciphering delay is bounded by s, such that C has b.d.d. in that case. □

In [25], we described a code which has a finite, but not bounded deciphering delay. This code is, however, not context-free, let alone 1tDB1C. Moreover, it was shown there that every regular code of finite deciphering delay also has bounded deciphering delay. As a corollary to Theorem 5.9 we discover even 1tDB1C languages which are codes of finite, but not bounded deciphering delay.

For the sake of simplicity we provide an example which does not refer to the machine construction in the proof of Theorem 5.9.

Example 5.10. Let $X := \{a, b\}$ and $C = a \cdot b^* a \cup \{ab^{n+1}ab^n a : n \in \mathbb{N}\} \cup \{b^{n+1}ab^n a : n \in \mathbb{N}\}$. Then $C \in$ 1tDB1C and is a code of finite but unbounded deciphering delay.

The last theorem proves the remaining undecidability result.

Theorem 5.11. *For languages $L \in$ 1tDB1C it is undecidable whether L is a code of finite (bounded) deciphering delay, even if we suppose L to be an ω-code.*

Proof. Simply alter the proof of Theorem 5.9 as follows. Define

$$L_0 := \{q_0\} \cup \{|^{5^t u}\# : t \in \mathbb{N} \text{ and } u \text{ is not divisible by 2, 3 or 5}\}\,.$$

Table 1. D stands for determinism, N for nondeterminism, d for decidable, u for undecidable and t for trivial; arrows indicate how (un)decidability results trivially propagate. When writing m-d.d. we assume $m \geq 1$.

Question	Condition	CF		LIN		1C		PBC		1tB1C		REG
		N	D	N	D	N	D	N	D	N	D	
code	-										$\longleftarrow$ u5.1	d
ω-code	code										$\longleftarrow$ u5.2	d
f.d.d.	code										$\longleftarrow$ u5.3	d
f.d.d.	ω-code										$\longleftarrow$ u5.11	d
b.d.d.	code										$\longleftarrow$ u5.3	d
b.d.d.	f.d.d.										$\longleftarrow$ u5.9	t
m-d.d.		$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	d4.2 $\longrightarrow$				
m-d.d.	(m+1)-d.d.				$\longleftarrow$ u5.8		$\longleftarrow$ u5.8	$\downarrow$				
Prefix	-	$\uparrow$	d	$\uparrow$	d	$\uparrow$	d					
		u	4.1	u	4.1	u	4.1					
Prefix	1-d.d.	5.6	$\downarrow$	5.6	$\downarrow$	5.6	$\downarrow$					
Suffix	-				$\uparrow$		$\uparrow$	d4.3 $\longrightarrow$				
Bifix	Prefix				$\longleftarrow$ u5.7		$\longleftarrow$ u5.7	$\downarrow$				
Bifix	-				$\downarrow$		$\downarrow$					

If $\mathfrak{M}$ does not halt, this may result in arbitrarily long sequences of code words $w_i, v_i \in L_1$ and $x \in L_0 \setminus \{q_0\}$ such that $q_0 x w_1 w_2 \ldots \sqsubseteq v_1 v_2 \ldots$ but no infinite sequence, because x gives a bound for its length. $\square$

In the preceding proof, the word $q_0 \in L_0$ has $\ell_{L_0}(q_0) = \infty$ regardless whether the code L_0 has f.d.d. or not. This shows that the converse of Property 3.5 is not valid.

6. Summary and prospects

Table 1 summarizes the results on the (un-)decidability status of code properties for various language classes. As regards the positive decidability results for regular languages, proofs can be found in Section 1.3 of [2] regarding the decidability of the code property, in [5] concerning the decidability of the ω-code and f.d.d. (or, here equivalently, the b.d.d.) property. Moreover, it was shown in [5, 25] that a regular code has f.d.d. iff it has b.d.d., so that one question becomes trivial.

It would be nice to know more about the (time or space) complexities of the decidable code properties; only the finite and regular code problems have received some attention until now [18, 24], although complexity questions have been explicitly raised in [3]. Let us remark as an example that the prefix code problem is just as hard as the emptiness problem for say (P)BC, since L is empty iff $\{a\} \cup \{aa\}L$, $a \in X$ is a prefix code, *cf.* also footnote 7. Moreover, the decision algorithm for DCF explained in Theorem 4.1 is much simpler than the algorithm for (P)BC given in Theorem 4.2 in terms of complexity.

Finally, there are lots of other code classes, see, *e.g.*, [13], for which it is still an open question to determine the borderline between the decidability and undecidability of the corresponding code class problems. Associated decidability questions are discussed in [23].

Acknowledgements. We gratefully acknowledge discussions with our colleagues U. Hertrampf, P. Kadau and R. Stiebe and the useful comments of the referees. A preliminary version of this paper appeared in *Developments in Language Theory DLT IV (Aachen 1999)*, edited by G. Rozenberg and W. Thomas; World Scientific, Singapore (2000) pp. 153–163.

References

[1] J. Berstel, *Transductions and Context-Free Languages*, *LAMM* **38**. Teubner, Stuttgart (1979).

[2] J. Berstel and D. Perrin. *Theory of Codes.* Pure and Applied Mathematics, Academic Press, Orlando (1985).

[3] J. Berstel and D. Perrin. Trends in the theory of codes. *EATCS Bull.* **29** (1986) 84–95.

[4] V. Bruyère. Automata and codes with bounded deciphering delay, in *Proc. LATIN'92*, edited by I. Simon. *Lect. Notes Comput. Sci.* **583** (1992).

[5] J. Devolder, M. Latteux, I. Litovski and L. Staiger, Codes and infinite words. *Acta Cybernetica* **11** (1994) 241–256.

[6] H. Fernau, IIFS and codes. In *Developments in Theoretical Computer Science*, edited by J. Dassow and A. Kelemenová. Gordon and Breach Science Publishers, Basel (1994) 141–152.

[7] H. Fernau and L. Staiger, IFS and control languages. *Inform. Comput.* **168** (2001) 125–143.

[8] S. Ginsburg, S. Greibach, and M. Harrison, One-way stack automata. *J. Assoc. Comput. Machinery* **14** (1967) 389–418.

[9] S. Greibach, Remarks on blind and partially blind one-way multicounter machines. *Theoret. Comput. Sci.* **7** (1978) 311–324.

[10] M.A. Harrison, *Introduction to Formal Language Theory.* Addison-Wesley series in computer science. Addison-Wesley, Reading (MA) (1978).

[11] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading (MA)(1979).

[12] H. Jürgensen, K. Salomaa and S. Yu, Transducers and the decidability of independence in free monoids. *Theoret. Comput. Sci.* **134** (1994) 107–117.

[13] H. Jürgensen and S. Konstantinidis, Codes. in *Handbook of Formal Languages, Volume I*, edited by G. Rozenberg and A. Salomaa. Springer, Berlin (1997) 511–607.

[14] S.R. Kosaraju, Decidability of reachability in vector addition systems, in *Proceedings 14th Annual ACM STOC* (1984) 267–281.

[15] K.-J. Lange and K. Reinhardt, Set automata. in *Combinatorics, Complexity and Logic, Proceedings of the DMTCS'96*, edited by D.S. Bridges. Springer, Berlin (1996) 321–329.

[16] V.I. Levenshtejn, Some properties of coding and self-adjusting automata for decoding messages (in Russian). *Problemy Kibernetiki* **11** (1964) 63–121. As regards translations, see [13], 604.

[17] R. Lindner and L. Staiger, *Algebraische Codierungstheorie; Theorie der sequentiellen Codierungen*, 11 *Elektronisches Rechnen und Regeln.* Akademie-Verlag, Berlin (1977).

[18] B.E. Litow, Parallel complexity of the regular code problem. *Inform. Comput.* **86** (1990) 107–114.

[19] E. Mayr, An algorithm for the general Petri net reachability problem. *SIAM J. Comput.* **13** (1984) 441–459.

[20] M.L. Minsky, Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Ann. Math.* **74** (1961) 437–455.

[21] M.L. Minsky, *Computation: Finite and Infinite Machines.* Prentice-Hall (1971).

[22] K. Reinhardt, *Prioritätszählerautomaten und die Synchronisation von Halbspursprachen.* Dissertation, Institut für Informatik, Universität Stuttgart (1994).

[23] A. Restivo, Codes and automata, in *Formal Properties of Finite Automata and Applications*, edited by J.E. Pin. *Lect. Notes Comput. Sci* **386** (1988) 186–198.

[24] W. Rytter, The space complexity of the unique decipherability problem. *Inform. Process. Lett.* **23** (1986) 1–3.

[25] L. Staiger, On infinitary finite length codes. *RAIRO-Theor. Inf. Appl.* **20** (1986) 483–494.

[26] L. Staiger, Codes, simplifying words, and open set condition. *Inform. Process. Lett.* **58** (1996) 297–301.

Communicated by H. Jan Hoogeboom.
Received March 17, 2003. Accepted May 10, 2005.