Computation with Absolutely No Space Overhead

Lane A. Hemaspaandra*
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
lane@cs.rochester.edu

Proshanto Mukherji
Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
mukherji@cs.rochester.edu

Till Tantau†
Fakultät für Elektrotechnik und Informatik
Technische Universität Berlin
10623 Berlin, Germany
tantau@cs.tu-berlin.de

Univ. of Rochester Comp. Sci. Dept. Technical Report TR-2002-779
May 15, 2002

Abstract

We study Turing machines that are allowed absolutely no space overhead. The only work space the machines have, beyond the fixed amount of memory implicit in their finite-state control, is that which they can create by cannibalizing the input bits' own space. This model more closely reflects the fixed-sized memory of real computers than does the standard complexity-theoretic model of linear space. Though some context-sensitive languages cannot be accepted by such machines, we show that a large subclasses of the context-free languages can even be accepted in polynomial time with absolutely no space overhead.

Keywords. Space overhead, space reuse, overhead-free computation, context-sensitive languages, linear space, deterministic linear languages, metalinear languages, hierarchy theorems.

1 Introduction

While recursion theory studies which problems can be solved *in principle* on a computer, complexity theory focuses on which problems can be solved *realistically.* Since it depends on context what resources are deemed "realistic," different resource bounds on various models

*Supported in part by grants NSF-CCR-9322513 and NSF-INT-9815095 /DAAD-315-PPP-gü-ab.
†Work done in part while visiting the University of Rochester, supported by a TU Berlin Erwin-Stephan-Prize grant.

have been studied. For example, deterministic linear space can be seen as a formalization of the limited memory of computers. Unfortunately, the standard complexity-theoretic formalizations may be too "rough" in realistic contexts, as most have hidden constants tucked away inside their definitions. Polynomial-time algorithms with a time bound of n^{100} and linear-space algorithms that need one megabyte of extra memory per input bit will typically be unhelpful from a practical point of view.

In this paper we study a model that we believe more realistically captures the fixed-sized memory of computers. We use (non-)deterministic one-tape Turing machines that may both read and write their tape. Crucially, we require that the machine may only write on tape cells that are nonempty at the beginning of the computation, when the tape is initialized with the input string (with an unwritable left endmarker $\vdash$ to its immediate left and an unwritable right endmarker $\dashv$ to its immediate right). The head may neither move left of the left endmarker nor right of the right endmarker. All (and only) words over the input alphabet, which is typically $\{0, 1\}$, are allowed as input strings. Also crucially, we require that the machine may only write symbols drawn from the *input* alphabet.

Thus, in our model of overhead-free computation the input initially completely fills the machine's memory and no auxiliary space is available. The machine can create work space by "cannibalizing" the space occupied by its input. However, the price of doing so is that the overwritten parts will potentially be lost (unless stored elsewhere via overwriting other parts of the input or unless stored in the machine's finite-state control). Note that the machine is not allowed to cheat by using an enriched tape alphabet. Allowing such cheating would transform our model into one accepting exactly the (non-)deterministic linear-space languages. As just a few examples of the large literature on

Although deterministic overhead-free computation is a natural model, its nondeterministic counterpart might appear to be of only theoretical interest. After all, nondeterministic computations are hardly "realistic" even if they are overhead-free. However, nondeterministic computations are useful in understanding the inherent limitations of overhead-free computation. An example of such a limitation is the fact that some context-sensitive languages cannot be accepted by overhead-free machines—*not even by nondeterministic ones*.

The class of languages accepted by deterministic overhead-free machines will be denoted DOF, and its nondeterministic counterpart will be denoted NOF. Although these classes "realistically" limit the *space* resources, the underlying machines can still potentially run exponentially long before they decide whether to accept. We will also study which languages can be accepted *efficiently* by overhead-free machines, that is, in polynomial time. Let $\mathrm{DOF}_{\mathrm{poly}}$ denote the class of those languages in DOF that are accepted by overhead-free machines running in polynomial time, and let $\mathrm{NOF}_{\mathrm{poly}}$ denote the class of languages in NOF that are accepted by overhead-free machines running in polynomial time.

Previous work on machines with limited alphabet size mostly concerned the *limitations* of such machines. For example, machines somewhat similar to overhead-free machines have been studied in a note by Feldman and Owings [5], namely linear bounded automata with bounded alphabet size. The work of Feldman and Owings shows that DOF is a proper subset of DCSL, the class of all deterministic context-sensitive languages. The work of Seiferas [20]

shows that NOF is a proper subset of CSL, the class of all context-sensitive languages.

In this paper we are mostly concerned with the *power* of overhead-free machines. We show that all deterministic linear languages [17] are contained even in the most restrictive of our four classes, namely $\mathrm{DOF}_{\mathrm{poly}}$. An even larger class of context-free languages, the metalinear languages [2,19], is contained in $\mathrm{NOF}_{\mathrm{poly}}$. In both cases we give an explicit algorithm. As additional indicators of the power of overhead-free computation, we show that $\mathrm{DOF}_{\mathrm{poly}}$ contains non-context-free sets and we show, using a reencoding trick, that DOF even contains PSPACE-complete sets.

To anyone who might think, "your model, which allows no extra space, is unnatural compared with the standard, natural notion of linear space," we would reply that actually the standard models of the field such as linear space are somewhat unnatural in a way that jumps off the page to students, but that professors have grown so used to that we rarely think about the unnaturalness (though we know well why we tolerate it). In the standard models, given an input the machine magically gets larger and larger amounts of extra space, and students often point out that they have yet to see machines whose amount of extra memory grows based on the input. The model is unnatural—yet we study things within it for all the standard, reasonable reasons that we learned long ago and know, love, and teach. Our point is not that standard models or classes are bad, but rather that ours is a natural model.

In addition, we point out that our model has strong roots in the literature. There is a large body of work, dating back decades and active through the present day, on *in situ* and "in place"/"inplace" algorithms (the latter in some places taking on a term-of-art special meaning with respect to a specific task [1]). Though that work is, loosely speaking, interested in computing functions (transformations) with almost no overhead rather than, as we are, computing languages with almost no overhead, that existing body of work does suggest that allowing essentially no additional space is a natural notion, and that defining classes to capture the thusly computable sets is also natural. As just as few examples of the literature on in-place algorithms, we mention [3,4,16,6,15,1].

This paper is organized as follows. In Section 2, we review basic concepts and define the classes DOF, NOF, $\mathrm{DOF}_{\mathrm{poly}}$, and $\mathrm{NOF}_{\mathrm{poly}}$. In Section 3, we demonstrate the power of overhead-free computation by giving explicit algorithms for accepting the abovementioned subclasses of the context-free languages with absolutely no overhead. In Section 4, we discuss the limitations of overhead-free computation.

2 Definitions and Review of Basic Concepts

In this section we first review basic concepts that will be needed in later sections. We then define the four models of overhead-freeness studied in this paper.

Given two alphabets Σ and Γ, a *substitution* is a mapping $s\colon \Sigma \to \mathcal{P}(\Gamma^*)$ that assigns a language $s(a)$ to every symbol $a \in \Sigma$. A substitution is extended to words by $s(a_1 \dots a_n) := \{w_1 \dots w_n \mid w_i \in s(a_i)\}$ and to languages by $s(L) := \bigcup_{w \in L} s(w)$. A *homomorphism* is a mapping $h\colon \Sigma \to \Gamma^*$. A homomorphism is *isometric* if all words in the range of h have the

same length. A homomorphism is extended to words by $h(a_1 \dots a_n) := h(a_1) \dots h(a_n)$ and to languages by $h(L) := \{h(w) \mid w \in L\}$. The *inverse image* of a language L is defined by $h^{-1}(L) := \{w \mid h(w) \in L\}$.

Let DLINSPACE $:= \bigcup_{k>0}$ DSPACE$[kn]$. Let NLINSPACE $:= \bigcup_{k>0}$ NSPACE$[kn]$. DCFL denotes the class of all deterministic context-free languages [11]. CFL denotes the class of all context-free languages. CSL denotes the class of all deterministic context-sensitive languages. It is well-known that CSL = NLINSPACE. The class DCSL (the "deterministic context-sensitive languages") is by definition DLINSPACE. Note that it is not hard to see that DLINSPACE and NLINSPACE contain the languages accepted by deterministic, respectively nondeterministic, one-tape Turing machines that write on only the cells occupied by the input—but in the model (not ours) in which machines *are* allowed to write arbitrary symbols of a possibly large tape alphabet.

We next review different classes of *linear* languages. These have been studied extensively in the literature [2] and have applications in probabilistic finite automata theory [18]. They are defined in terms of the following special types of context-free grammars.

Definition 2.1 ([19]). A grammar is *linear* if it is a context-free grammar in which the right-hand sides of all productions contain at most one nonterminal.

Definition 2.2 ([19]). A context-free grammar $G = (N, T, S, P)$ is said to be a *k-linear grammar* if it has the form of a linear grammar plus one additional rule of the form $S \to S_1 S_2 \dots S_k$, where none of the S_i may appear on the right-hand side of any other rule and S may not appear in any other rule at all.

Definition 2.3 ([17,19]). A linear grammar $G = (N, T, S, P)$ is *deterministic linear* if it has the following two properties. First, all right-hand sides containing nonterminals have this nonterminal at the second position. Second, for every $X \in N$ and $a \in T$ there is at most one production with X on the left-hand side whose right-hand side starts with a.

A language is called *linear* (*k-linear*, *deterministic linear*) if it is generated by a grammar that is linear (k-linear, deterministic linear). It is called *metalinear* if it is k-linear for some k.

The above standard definition of deterministic linear grammars is the one given by Nasu and Honda in 1969 [17]. It is rather restrictive, which will make our proofs more transparent. The full power of deterministic linear languages can in fact be better appreciated by looking at the far more flexible-seeming definition—which includes a much broader range of grammars—given by Ibarra, Jiang, and Ravikumar in 1988 [12]. Crucially, Holzer and Lange [9] proved that the latter definition in fact yields exactly the same class of languages as the Nasu–Honda definition except it accepts one additional language: the language containing exactly the empty string. Since that pathological language is in fact in $\mathrm{DOF}_{\mathrm{poly}}$, all results of this paper hold under either definition. (Note: Holzer and Lange [9] have proposed a new, completely different, machine-based definition of what they feel should be called a "deterministic linear (context-free) language," but this proposed terminological change has not caught on, so we follow the standard, traditional terminology.)

To conclude this section, we now define the four different classes of overhead-free computation. To do this rigorously we must tackle one technical issue: The notion of overhead-freeness is sensible only if languages "carry around" their underlying alphabet. Normally, the difference between, say, the language $A = \{1^p \mid p \text{ is prime}\}$ taken over the alphabet $\{1\}$ and the same language A taken over the alphabet $\{0, 1\}$ is irrelevant in complexity theory, since we can enlarge the tape alphabet appropriately. In contrast with this, for our notion of overhead-freeness it certainly makes a difference whether the input alphabet is unary or binary, as a unary input alphabet rids us of the possibility of interestingly writing *anything* onto the tape, see Theorem 4.1. Thus, from a formal point of view we consider our complexity classes to consist of tuples (L, Σ) consisting of a language L and an alphabet Σ such that $L \subseteq \Sigma^*$.

For the following definition, recall that we called a machine *overhead-free* if it writes on only those cells that were initially filled with the input and if it writes symbols drawn only from the input alphabet.

Definition 2.4. A pair (L, Σ) is in the class DOF if L is accepted by a deterministic overhead-free machine with input alphabet Σ. A pair (L, Σ) is in $\mathrm{DOF}_{\mathrm{poly}}$ if L is accepted by a deterministic overhead-free, polynomial-time machine with input alphabet Σ. The counterparts to DOF and $\mathrm{DOF}_{\mathrm{poly}}$ defined in terms of nondeterministic machines are denoted NOF and $\mathrm{NOF}_{\mathrm{poly}}$.

As differing input alphabets are mainly a technical subtlety, we will in the following speak just of L when the Σ is clear from context.

3 The Power of Overhead-Free Computation

In this section we demonstrate the power of overhead-free computation. We first give an explicit example of a non-context-free set that nonetheless is in the smallest of our classes, namely $\mathrm{DOF}_{\mathrm{poly}}$. Then we show that DOF contains a PSPACE-complete set. The final part of this section is taken up by a sequence of theorems that establish containment in $\mathrm{DOF}_{\mathrm{poly}}$, or at least $\mathrm{NOF}_{\mathrm{poly}}$, for larger and larger classes.

As an introductory example of overhead-free computation, we show that the language $A := \{0^n1^n0^n \mid n \geq 1\}$ is in $\mathrm{DOF}_{\mathrm{poly}}$. Since this set is not context-free, $\mathrm{DOF}_{\mathrm{poly}}$ contains non-context-free sets.

Theorem 3.1. *There is a set in* $\mathrm{DOF}_{\mathrm{poly}}$ *that is not context-free.*

Proof. We show $A = \{0^n1^n0^n \mid n \geq 1\} \in \mathrm{DOF}_{\mathrm{poly}}$ via a machine M. On input $w \in \{0, 1\}^*$, using a left-to-right sweep, M first ensures that the input is of the form $0^+1^+0^+$. Then it moves back to the left end. During the following computation the tape's content will always be of the form $0^*1^*0^*1^*$. The machine now loops through the following instructions: In the first 0's block replace the last 0 by a 1, which enlarges the following 1's block by one 1. In this following block replace the last two 1's by 0's, which enlarges the following 0's block by two 0's. In this block replace the last three 0's by 1's. If you inadvertently hit the right

endmarker at any point, reject. Return to the left end and check whether the tape solely consists of 1's. If so, accept, otherwise repeat.

Clearly, the computation is overhead-free and runs in polynomial time. Furthermore, the input word will be accepted exactly if it is of the prescribed form. □

Our next aim is to show that DOF contains a PSPACE-complete set. To prove this, we first show that given any language $L \in$ DLINSPACE we can find a closely related language $L' \subseteq \{0,1\}^*$ that is accepted by an overhead-free machine.

Lemma 3.2. *Let $L \in$ DLINSPACE with $L \subseteq \Sigma^*$. Then there exists an isometric homomorphism $h\colon \Sigma \to \{0,1\}^*$ such that for $L' := h(L) \subseteq \{0,1\}^*$ we have $L' \in$ DOF. Similarly, if $L \in$ NLINSPACE there exists an isometric homomorphism h such that $L' := h(L) \in$ NOF.*

Proof. Let $L \in$ DLINSPACE via a machine M that never writes on any cells other than those that contained the input. As mentioned earlier, every language in DLINSPACE can be accepted in such a fashion. Let M use the tape alphabet $\Gamma \supseteq \Sigma$, which may be strictly richer than Σ. Let $k := \lceil \log_2 |\Gamma| \rceil$. Then there exists an injective mapping $g\colon \Gamma \to \{0,1\}^k$, which codes every symbol in Γ as a binary string of length k. Let h be the restriction of g to the domain Σ.

We now show $h(L) \in$ DOF via a machine M'. This machine always reads k symbols as a block. On input w it first ensures that w consists only of blocks that encode symbols from $\Sigma \subseteq \Gamma$. Then $w = g(u_1)g(u_2)\dots g(u_m)$ for some sequence of u_i's in Σ. For convenience, we will overload g to operate on strings as well as symbols when our meaning is clear from context. Thus here $w = g(u)$ for the string $u = u_1u_2\dots u_m$. M' now returns to the beginning of the input and starts a simulation of what M would do on input u. Every time M reads/writes a single symbol, M' reads/writes a block of size k that encodes the read/written symbol. This way, whenever the machine M would reach a state q with tape content $u' \in \Gamma^*$, the machine M' will also reach state q with the tape content $g(u') \in \{0,1\}^*$. If M would accept, M' does.

For the nondeterministic case the simulation works analogously. □

Since DLINSPACE and NLINSPACE are clearly closed under inverse isometric homomorphisms, we have the following corollary.

Corollary 3.3. *The closure of* DOF *under inverse isometric homomorphisms is exactly* DLINSPACE*. Likewise, the closure of* NOF *under inverse isometric homomorphisms is exactly* NLINSPACE*.*

Theorem 3.4. *The class* DOF *contains a $\leq_{\mathrm{m}}^{\log}$-complete set for* PSPACE*.*

Proof. Take any $\leq_{\mathrm{m}}^{\log}$-complete problem $A \in$ PSPACE. This problem $\leq_{\mathrm{m}}^{\log}$-reduces via standard padding techniques to a problem B in DLINSPACE. But B can be inverse-isometric-homomorphism-mapped (and thus certainly $\leq_{\mathrm{m}}^{\log}$-reduced) to a set $C \in$ DOF by Corollary 3.3. This set C is $\leq_{\mathrm{m}}^{\log}$-complete for PSPACE. □

The fact that powerful sets can reduce to DOF does not say that those sets are in DOF themselves. In fact, since DOF is obviously a subset of DLINSPACE, by the Space Hierarchy Theorem some PSPACE languages are not in DOF. For example there is a language in DSPACE[n^2] that is not in DOF.

The rest of this section is devoted to proving the containment in $\mathrm{DOF}_{\mathrm{poly}}$ or $\mathrm{NOF}_{\mathrm{poly}}$ of ever larger subclasses of CFL. We start with the simplest case, the regular languages.

Theorem 3.5. *All regular languages are in* $\mathrm{DOF}_{\mathrm{poly}}$.

Proof. Regular languages are even in $\mathrm{DOF}_{\mathrm{linear}}$ via machines that move their heads steadily to the right and never write at all. □

Our next aim is to prove that all deterministic linear languages (see Section 2 for a detailed definition) are in $\mathrm{DOF}_{\mathrm{poly}}$. We start with a useful lemma on the effect of a *constant* amount of additional space at the beginning or end of the input on overhead-free computation. Roughly put, it has no effect.

Lemma 3.6. *Let* $C \in \{\mathrm{DOF}, \mathrm{NOF}, \mathrm{DOF}_{\mathrm{poly}}, \mathrm{NOF}_{\mathrm{poly}}\}$*, let* L *be a language over the alphabet* Σ*, and let* $u, v \in \Sigma^*$*. Then* $\{uzv \mid z \in L\} \in C$ *iff* $L \in C$*.*

Proof. Let $L \in C$ via M. To show $\{uzv \mid z \in L\} \in C$, on input w we first check whether $w = uzv$ for some word $z \in \Sigma^*$. If so, we simulate M on input z. During this simulation we pretend to see a simulated left endmarker whenever we are actually $|u|$ many symbols from the real left end, and to see a simulated right endmarker whenever we are $|v|$ many symbols from the real right end. We do so by, between each simulated step, moving up to $|u|$ extra squares left and up to $|v|$ extra squares right to detect whether we are thusly close to the real ends.

For the only-if part, let $\{uzv \mid z \in L\} \in C$ via machine M. Then $L \in C$ via the machine M' that, essentially, simulates M but when over the "u" or "v" parts (which do not exist in its own input) uses its finite-state control to keep track of its head location, and that globally keeps in its finite-state control the content currently on those hypothetical cells. (By "using its finite-state control to keep track of" we refer to the standard fact that given a machine with state set F we can build a machine with state set $F' := F \times \{0,1\}^k$, thus in effect adding a constant number of usable extra "memory" bits to our finite control.) □

Theorem 3.7. *All deterministic linear languages belong to* $\mathrm{DOF}_{\mathrm{poly}}$.

Proof. Let L be a deterministic linear language generated by a grammar $G = (N, T, S, P)$. All deterministic linear *tally* languages are regular, since in the right-hand side of every rule we can shift an existing nonterminal symbol to the end without changing the generated language. Thus we need only consider the case that T contains at least two distinct symbols. We will refer to two such symbols as 0 and 1.

By Lemma 3.6 it suffices to prove that $\{1z1 \mid z \in L\} \in \mathrm{DOF}_{\mathrm{poly}}$ via some machine M. Let w be an input word. We first check whether $w = 1c_1 \ldots c_n 1$ with $c_i \in T$. We must now check whether $z := c_1 \ldots c_n \in L$, that is, whether there exists a derivation of z in G.

Since the grammar is deterministic linear, it holds that if a derivation of z is possible at all, then there exists exactly one possible derivation and it has the form

$$S = l_0 X_0 r_0 \Rightarrow_G l_1 X_1 r_1 \Rightarrow_G l_2 X_2 r_2 \Rightarrow_G \cdots \Rightarrow_G l_t X_t r_t \Rightarrow_G z,$$

where the X_i are nonterminals and the l_i and r_i are words over T. Note that each l_i is a prefix of z and each r_i is a suffix of z. Let $\lambda_i := |l_i|$ and $\rho_i := |r_i|$ be the lengths of l_i and r_i, respectively.

The computation of M consists of a big loop. In each iteration of the loop the machine checks one step of the derivation. When the step $l_i X_i r_i \Rightarrow l_{i+1} X_{i+1} r_{i+1}$ of the derivation is checked, the machine has the symbol X_i stored in its finite-state control. The tape has the form $0^{\lambda_i} 1 c_{\lambda_i+1} \dots c_{n-\rho_i-1} 1 0^{\rho_i}$, that is, all of l_i (and r_i) will already have been replaced by 0's, followed (respectively, preceded) by 1's, which serve as stop markers. The machine's goal is to check whether $X_i \Rightarrow^* c_{\lambda_i+1} \dots c_{n-\rho_i-1}$. Note that, indeed, in the zeroth step the tape has exactly the prescribed form.

The machine M now starts scanning from the left end until it hits a 1. Then it moves forward one symbol and "has a look" at the symbol c_{λ_i+1}. By definition there can be at most one rule with a left-hand side X_i whose right-hand side starts with c_{λ_i+1}. Let $X_i \to c_{\lambda_i+1} X_{i+1} y$ be this rule. The machine now moves to the end of the input and scans back from there to the first 1 from the right. Having found this stop marker, it can check whether $c_{n-\rho_i-|y|} \dots c_{n-\rho_i-1} = y$. If so, the machine has verified that the derivation step $l_i X_i r_i \Rightarrow l_{i+1} X_{i+1} r_{i+1}$ was permissible. All that remains to be done is to move the left and right stop markers inward by an appropriate number of cells.

In the last step, when the machine notices that the nonterminal symbol X_t is replaced by a word $a \in T^*$, it verifies that the tape's content is $0^* 1 a 1 0^*$. If this is the case the machine accepts, and otherwise it rejects. As the computation is deterministic, overhead-free, and polynomially time-bounded, we have the claim. □

Our final aim for this section is to show that all metalinear languages can be accepted by nondeterministic overhead-free machines. To prove this, we show the following stronger theorem first.

Theorem 3.8. *Let $A \subseteq \Delta^*$ be a regular language and let $s\colon \Delta \to \mathcal{P}(\Sigma^*)$ be a substitution such that for all $d \in \Delta$ the language $s(d)$ is linear. Then $s(A) \in \mathrm{NOF}_{\mathrm{poly}}$.*

Proof. By definition we have $z \in s(A)$ iff there exists a word $y = y_1 \dots y_k \in A$ and words $z_1, \dots, z_k \in \Sigma^*$ such that $z = z_1 \dots z_k$ and $z_i \in s(y_i)$ for all $i \in \{1, \dots, k\}$. Let D be a deterministic finite automaton that accepts A. In the following we first show $s(A) \in \mathrm{NOF}$ via some machine M. Later on we will add some safe-guards to ensure that M runs in polynomial time. This will prove $s(A) \in \mathrm{NOF}_{\mathrm{poly}}$.

We now give a rough sketch of M's behavior. On input z it starts a main loop. In this main loop the machine guesses a word $y_1 \dots y_k \in A$, that is, for every word in A there is a nondeterministic path on which this word is guessed. Next, for each symbol y_i the machine guesses a word $z_i \in s(y_i)$ in a sub-loop and compares this word with an appropriate part of

the input. At the end, if the machine has verified that the input z can be decomposed in the form $z = z_1 \dots z_k$ with $z_i \in s(y_i)$ for all $i \in \{1, \dots, k\}$, it accepts. What remains to be shown is how the machine avoids producing any overhead.

As in the previous proof, we may assume that there exist two distinct symbols $0, 1 \in \Sigma$. We show $\{1z \mid z \in L\} \in \mathrm{NOF}_{\mathrm{poly}}$. Let $w = 1z = 1c_1 \dots c_n$ be given as input. For $d \in \Delta$, let $G_d = (N_d, \Sigma, S_d, P_d)$ denote a linear grammar that generates $s(d)$.

In the main loop the machine nondeterministically guesses a word $y \in A$ by guessing an accepting computation of the automaton D. However, this word y is not written down anywhere. Rather, the machine stores just the current symbol $d := y_i$ and the current state of the automaton D in its finite-state control (see the related comments in the proof of Lemma 3.6). When the i'th symbol is guessed, the machine will already have verified that z starts with $z_1 \dots z_{i-1}$, with $z_j \in s(y_j)$ for $j \in \{1, \dots, i-1\}$. At this point the content of the tape will be $0^\ell 1 c_{\ell+1} \dots c_n$, where $\ell = |z_1 \dots z_{i-1}|$. In other words, the $z_1 \dots z_{i-1}$ will have been replaced by 0's followed by a stop marker. We will call this the "land" stop marker as it marks the end of the "land" $0^\ell 1$ before the rough "sea" $c_{\ell+1} \dots c_n$.

The tricky part is verifying that after the land stop marker comes a z_i for which there exists a derivation

$$S_d = l_0 X_0 r_0 \Rightarrow_{G_d} l_1 X_1 r_1 \Rightarrow_{G_d} l_2 X_2 r_2 \Rightarrow_{G_d} \cdots \Rightarrow_{G_d} l_t X_t r_t \Rightarrow_{G_d} z_i.$$

As the word z_i typically ends somewhere in the middle of the tape, we cannot employ the "eat away from the outside" algorithm from the previous proof. Instead, we now "eat away from the inside."

Starting from the land stop marker we move right until we nondeterministically guess that we have reached the point where the last rule $X_t \to a$ with $a \in T^*$ was employed. We check whether we really find a at that position. If so, we replace a by $10^{|a|-2}1$ (if $|a| < 2$ we defer this replacement until we have gathered enough space in the following steps). The tape's content will now be

$$0^\ell 1 c_{\ell+1} \dots c_{\ell+|l_t|} 10^{|a|-2} 1 c_{\ell+|l_t|+|a|+1} \dots c_n$$

and the head will be inside the "island" of 0's in the middle. Next, the machine nondeterministically guesses which rule $X_{t-1} \Rightarrow_{G_d} y X_t y'$ was employed in the last step of the derivation of z_i. It then checks whether it finds y immediately left of the left island stop marker and y' immediately right of the right island stop marker. If so, it pushes the left island stop marker $|y|$ many cells to the left and the right island stop marker $|y'|$ many cells to the right.

Note that the machine will *not notice* if it inadvertently pushes the left island stop marker over the land stop marker—after all, 0 and 1 are perfectly legitimate input symbols. However, the machine will notice such a mistake later on, when it has eaten away a complete derivation of z_i. At this point, if all went well, the land stop marker must be directly adjacent to the left island stop marker. The machine must hence check whether the tape left of the island (whose boundary the machine knows) looks like this: 0^*1. If this is the case, the land stop marker can be pushed to the right end of the island and the next stage can be entered.

When the machine has guessed a complete word $y \in A$, it can check whether the land stop marker has hit the right end. If so, it accepts.

It remains to show how we can ensure that M runs in polynomial time. There are two places where M might "spend too much time." First, in a derivation of a word z_i rules of the form $X \to Y$ with $X, Y \in N_d$ might be applied over and over again, that is, a nonterminal might repeatedly be replaced by a nonterminal without any terminals being read. Since every row of more than $\|N_d\|$ many such replacements must necessarily contain the same nonterminal twice, no derivation of a word z_i *needs* such long rows of replacement. Thus, we keep a counter of how often we applied rules of the form $X \to Y$ in a row. We reject if this counter exceeds the cardinality of N_d. Since this cardinality is fixed, we can store this counter in our finite control.

The second place where M might loop is a long row of z_i's that are all the empty word. In this case M constantly switches the stored internal state and the output d of the automaton D that accepts A, but always guesses that the empty word is derived from S_d. Similarly to first case, if we make more internal switches in a row than there are states in D, we must visit the same state twice. Thus we do not really *need* more empty derivations in a row than there are states in D. We keep a counter of the number of times in a row the empty string was substituted for z_i. If this counter exceeds the number of states in D, we reject. This counter can also be kept in our finite control. □

Note that there are three points in the above proof where we use nondeterminism. First, we nondeterministically guess the word $y \in A$. Second, we nondeterministically guess the "middle" of the z_i's, where their derivations end. Third, we nondeterministically guess the derivations of the z_i's themselves.

Corollary 3.9. *All metalinear languages belong to* $\mathrm{NOF}_{\mathrm{poly}}$.

Proof. Every k-linear language L can be written as $L = s(A)$, where A contains just one word $a_1 \dots a_k$ consisting of k different symbols, and s is a substitution that assigns a linear language L_i to each a_i. □

4 Limitations of Overhead-Free Computation

The previous section demonstrated that several interesting languages can be accepted by overhead-free machines. In this section we discuss what cannot be done using overhead-free machines.

We begin this section with a theorem that shows that overhead-free machines on unary alphabets are just as powerless (or powerful, depending on your point of view) as finite automata. Theorems 4.2 and 4.3 then show that there are (non-)deterministic context-sensitive language that cannot be accepted by (non-)deterministic overhead-free machines. Both results are based on diagonalization techniques. In the rest of the section we discuss whether certain *natural* sets can be accepted in an overhead-free fashion.

Theorem 4.1. *Let L be a tally set. Then the following are equivalent: (a) L is regular, (b) $L \in \mathrm{DOF}_{\mathrm{poly}}$, (c) $L \in \mathrm{DOF}$, (d) $L \in \mathrm{NOF}_{\mathrm{poly}}$, and (e) $L \in \mathrm{NOF}$.*

Proof. If L is regular, then by Theorem 3.5 it is in $\mathrm{DOF}_{\mathrm{poly}}$ and hence also in all of the other three classes. For the other direction, let $L \in \mathrm{NOF}$ via an overhead-free machine M. Since the input alphabet is unary, M behaves exactly like a nondeterministic finite two-way automaton. As is well-known, nondeterministic finite two-way automata accept only regular sets. □

Theorem 4.2. $\mathrm{DOF} \subsetneq \mathrm{DLINSPACE}$.

Proof. This follows immediately from Corollary 2 of a paper by Feldman and Owings [5]. They show that for every constant m deterministic linear-bounded automata with alphabet size at most m cannot accept all deterministic context-sensitive languages. □

Theorem 4.3. $\mathrm{NOF} \subsetneq \mathrm{NLINSPACE}$.

Proof. Seiferas [20] has shown that for every m there exists a language in NLINSPACE that cannot be accepted by any nondeterministic off-line Turing machine, that uses only m different symbols on its tape. Since any overhead-free machine can be simulated by a linear space off-line machine that first copies its input to its work tape, we get the claim. □

An alternative proof of Theorem 4.3 can be based on combining Corollary 1 of the paper of Feldman and Owings [5] with the Immerman-Szelepcsényi theorem [13,21]. The corollary states that for each m there is a language whose complement is context-sensitive and that cannot be accepted by a nondeterministic linear-bounded automaton whose alphabet size is bounded by m. This is another example of the often encountered effect that the Immerman-Szelepcsényi technique can be used to simplify nondeterministic space hierarchy proofs (see [13,7]).

Theorems 4.2 and 4.3 show that overhead-free computation is less powerful than linear-space computation *in principle*. A next step would be to prove that certain simple, natural context-sensitive languages cannot be accepted in an overhead-free fashion. Our candidate for a language that is not in NOF is $L_1 := \{ww \mid w \in \{0,1\}^*\}$. Our candidate for a language that is not in DOF is $L_2 := \{uu^{-1}vv^{-1} \mid u, v \in \{0,1\}^*\}$. Since L_2 is metalinear, proving $L_2 \notin \mathrm{DOF}$ would be especially satisfying as it would also separate nondeterministic and deterministic overhead-free computation.

Interestingly, the languages L_1 and L_2, though we name them candidate non-NOF (respectively non-DOF) languages, are in "2-head-$\mathrm{DOF}_{\mathrm{poly}}$," the analog of $\mathrm{DOF}_{\mathrm{poly}}$ in which our overhead-free machine has two heads. For L_1 this is very easily seen. For L_2 the algorithm works as follows: In a big loop, head 1 iterates over all symbols of the input, while head 2 stays at the left endmarker. The body of the main loop consists of two checks, in which the machine tries to verify whether the subwords before and after head 1 are palindromes. For the first check, head 2 is moved right and head 1 is moved left until head 1 hits the left endmarker. Then head 1 is moved to the right endmarker. For the second check, head 1 is moved left and head 2 is moved right until head 2 hits the right

endmarker. Then head 2 is moved back to the left endmarker once more. If at any stage of either check the symbols under the heads differ, the checks will be said to "fail," but they are completed nevertheless. If the subwords before and after head 1 are palindromes, the checks will not "fail" and the machine can accept its input. In any case, the heads will have resumed their previous at the end of each iteration of the loop.

These observations raise the question of how powerful extra heads make our model. First, by a classic result of Hartmanis [8], even $\mathcal{O}(1)$-head *finite automata* taken collectively yield the power of logarithmic-space Turing computation. Thus, at least in that different context, additional heads are well-known to be a very powerful resource. However, using the same argument as in Theorem 4.3, Seiferas' results [20] can be used to show that for every m there exists a context-sensitive language that is not in m-head-NOF.

5 Conclusion

In this paper we introduced a new computational model, namely overhead-free computation, in which Turing machines are allowed to manipulate their input, but may not use any symbols other than the input alphabet. For the case of a binary input alphabet this models what a personal computer or, perhaps more interestingly, a smart-card can compute in its fixed-sized non-control memory, if the input initially fills the whole non-control memory. Building on this model we defined the four complexity classes DOF, NOF, $\mathrm{DOF}_{\mathrm{poly}}$, and $\mathrm{NOF}_{\mathrm{poly}}$ and studied how these classes relate to standard formal-language/complexity classes. The most "realistic" of the four classes is $\mathrm{DOF}_{\mathrm{poly}}$, which contains the languages that can be accepted efficiently (that is, in polynomial time) with absolutely no space overhead. We showed by means of an explicit overhead-free algorithm that an important class of languages, namely the class of deterministic linear languages, is a subset of $\mathrm{DOF}_{\mathrm{poly}}$. For the larger class of metalinear languages we proved containment in $\mathrm{NOF}_{\mathrm{poly}}$.

These results suggest that $\mathrm{CFL} \subseteq \mathrm{NOF}$ might hold. Our research gave neither proof nor disproof of this inclusion. We recommend further research to characterize exactly those context-free grammars that generate languages in NOF. Naturally, the ultimate goal of this line of research would be to prove $\mathrm{CFL} \subseteq \mathrm{DOF}_{\mathrm{poly}}$, which would be a major improvement of the well-known $\mathrm{CFL} \subseteq \mathrm{P}$ result due to Cocke, Younger, and Kasami [14, 22], or to prove the lack of that and other inclusions.

It is known (see [10, Section 11.3]) that some context-free languages inherently need logarithmic space on deterministic off-line Turing machines and that some context-free languages inherently need linear space on deterministic on-line Turing machines. However, these results do not imply that these languages are not in NOF, as both in the on-line and off-line models of [10] the input is read-only. In contrast, in our model the input *may* be overwritten, albeit at the cost of losing the overwritten input bits. In fact, the very set used in the logspace-overhead space lower bound of [10] can be shown to belong to $\mathrm{DOF}_{\mathrm{poly}}$.

Acknowledgments

We thank Edith Hemaspaandra, Mitsunori Ogihara, and Jonathan Shaw for helpful discussions.

References

[1] A. Amir, G. Landau, and D. Sokol. Inplace run-length 2D compressed search. *Theoretical Computer Science*, 2002. To appear.

[2] N. Chomsky and M. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North Holland, Amsterdam, 1963.

[3] E. Dijkstra. Smoothsort, an alternative for sorting in situ. *Science of Computer Programming*, 1(3):223–233, 1982.

[4] E. Dijkstra and A. van Gastern. An introduction to three algorithms for sorting in situ. *Information Processing Letters*, 15(3):129–134, 1982.

[5] E. Feldman and J. Owings, Jr. A class of universal linear bounded automata. *Information Sciences*, 6:187–190, 1973.

[6] K. Fishkin. Performing in-place affine transformations in constant space. In *Proceedings of Graphics Interface '92*, pages 106–114, 1992.

[7] J. Geske. Nondeterminism, bi-immunity and almost-everywhere complexity. *IEICE Transactions on Communications, Electronics, Information, and Systems*, E76, 1993.

[8] J. Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1:336–344, 1972.

[9] M. Holzer and K. Lange. On the complexities of linear LL(1) and LR(1) grammars. In *Proceedings of the 9th Conference on Fundamentals of Computation Theory*, pages 299–308. Springer Verlag *Lecture Notes in Computer Science #710*, August 1993.

[10] J. Hopcroft and J. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1969.

[11] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[12] O. Ibarra, T. Jiang, and B. Ravikumar. Some subclasses of context-free languages in NC^1. *Information Processing Letters*, 29(3):111–117, 1988.

[13] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[14] T. Kasami. An efficient recognition and syntax algorithm for context-free languages. Scientific Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Mass., 1965.

[15] J. Katajainen and T. Pasanen. In-place sorting with fewer moves. *Information Processing Letters*, 70:31–37, 1999.

[16] H. Mannila and E. Ukkonen. A simple linear-time algorithm for in situ merging. *Information Processing Letters*, 18(4):203–208, 1984.

[17] M. Nasu and N. Honda. Mappings induced by PGSM-mappings and some recursively unsolvable problems of finite probabilistic automata. *Information and Control*, 15(3):250–273, 1969.

[18] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, New York, 1971.

[19] A. Salomaa. *Formal Languages*. Academic Press, 1973.

[20] J. Seiferas. Relating refined space complexity classes. *Journal of Computer and System Sciences*, 14:100–129, 1977.

[21] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[22] D. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.