Between SC and LOGDCFL: families of languages accepted by logarithmic-space deterministic auxiliary depth-k storage automata

Tomoyuki Yamakami
Faculty of Engineering, University of Fukui, Fukui, Japan

ABSTRACT
The closure of deterministic context-free languages under logarithmic-space many-one reductions (L-m-reductions), known as LOGDCFL, has been studied in depth from an aspect of parallel computability because it is nicely situated between L and AC¹ ∩ SC². By replacing a memory device of pushdown automata with an access-controlled storage tape, we introduce a computational model of one-way deterministic depth-k storage automata (k-sda’s) whose tape cells are freely modified during the first k accesses and then become blank forever. These k-sda’s naturally induce the language family kSDA. Similarly to LOGDCFL, we study the closure LOGkSDA of all languages in kSDA under L-m-reductions. We demonstrate that DCFL ⊆ kSDA ⊆ SC by significantly extending Cook’s early result (1979) of DCFL ⊆ SC. The entire hierarch of LOGkSDA for all k ≥ 1 therefore lies between LOGDCFL and SC. As an immediate consequence, we obtain the same simulation bounds for Hibbard’s limited automata. We further characterize LOGkSDA in terms of a new machine model, called logarithmic-space deterministic auxiliary depth-k storage automata that run in polynomial time. These machines are as powerful as a polynomial-time two-way multi-head deterministic depth-k storage automata. We also provide a ‘generic’ LOGkSDA-complete language under L-m-reductions by constructing a two-way universal simulator working for all k-sda’s.

1. DCFL, LOGDCFL, and beyond

In the literature, numerous computational models and associated language families have been proposed to capture various aspects of parallel computation. Of those language families, we wish to pay special attention to the family known as LOGDCFL, which is obtained from DCFL, the family of all deterministic context-free (dcf) languages, by taking the closure under logarithmic-space many-one reductions (or L-m-reductions, for short) [3, 23]. These dcf languages were first defined in 1966 by Ginsburg and Greibach [8] and their fundamental properties were studied intensively since then. It is well known that DCFL is a proper subfamily of CFL, the family of all context-free languages, because the context-free language \{ww^R \mid w \in \{0, 1\}^*\} (where w^R means the reverse of w), for instance, does not belong to DCFL. The dcf languages in general behave quite differently...
from the context-free languages. As an example of such differences, DCFL is closed under complementation, while CFL is not. This fact structurally distinguishes between DCFL and CFL. Moreover, dcf languages require computational resources of polynomial time and $O(\log^2 n)$ space simultaneously [4]; however, we do not know the same statement holds for context-free languages. Although dcf languages are accepted by one-way deterministic pushdown automata (or 1dpda’s), these languages have a close connection to highly parallelized computation because of the nice inclusions $NC^1 \subseteq L \subseteq NL \subseteq LOGCFL \subseteq AC^1$ and $L \subseteq LOGDCFL \subseteq SC^2$, and thus LOGDCFL has played a key role in discussing parallel complexity issues within $P$.

It is known that LOGDCFL can be characterized without using L-m-reductions by several other intriguing machine models, which include: Cook’s polynomial-time logarithmic-space deterministic auxiliary pushdown automata [3], two-way multi-head deterministic pushdown automata running in polynomial time [23], logarithmic-time CROW-PRAMs with polynomially many processors [5], and circuits made up of polynomially many multiplex select gates having logarithmic depth [6] or having polynomial proof-tree size [18]. Such a variety of characterizations prove LOGDCFL to be a robust and fully-applicable notion in computer science.

Another important feature of LOGDCFL (as well as its underlying DCFL) is the existence of ‘complete’ languages, which are practically the most difficult languages in DCFL to recognize. Notice that a language $L$ is said to be L-m-complete for a family $\mathcal{F}$ of languages (or $L$ is $\mathcal{F}$-complete, for short) if $L$ belongs to $\mathcal{F}$ and every language in $\mathcal{F}$ is L-m-reducible to $L$. Sudborough [23] first constructed such a language, called $L^{(m)}_0$, which possesses the highest complexity (which he quoted as ‘tape hardest’) among all dcf languages under L-m-reductions; therefore, $L^{(m)}_0$ is L-m-complete for DCFL and also for LOGDCFL. Using Sudborough’s hardest languages, Lohrey [17] lately presented another LOGDCFL-complete problem based on semi-Thue systems. Nonetheless, only a few languages are known today to be complete for DCFL as well as LOGDCFL.

A large void seems to lie between DCFL and CFL (as well as co-CFL). This void has been filled with, for example, the union hierarchy \{DCFL[k] \mid k \geq 1\} and the intersection hierarchy \{DCFL(k) \mid k \geq 1\} over DCFL, where DCFL[k] (resp., DCFL(k)) is composed of all unions (resp., intersections) of $k$ dcf languages. They truly form distinctive infinite hierarchies [16, 26]. Taking a quite different approach, Hibbard [12] devised specific rewriting systems, known as deterministic scan limited automata. Those rewriting system were lately remodelled in [20, 21] as single input/storage-tape 2-way deterministic linear-bounded automata that can modify the contents of their tape cells whenever the associated tape heads access the tape cells (when a tape head makes a turn, however, we count it twice); however, such modifications are limited to only the first $k$ accesses and then the tape cells are frozen forever. Those machines are dubbed as deterministic $k$-limited automata (or $k$-lda’s, for short). Numerous followup studies, including Pighizzini and Prigioniero [22], Kutrib and Wendlandt [15], Kutrib, Pighizzini, and Wendlandt [14], and Yamakami [25], have lately revitalized an old study on $k$-lda’s. It is possible to impose on $k$-lda’s the so-called blank-skipping property [25], by which inner states of $k$-sda’s cannot be changed while reading any blank symbol. A drawback of Hibbard’s model is that the use of a single tape prohibits us from accessing input and memory simultaneously. Seemingly, this drawback makes it difficult to construct a language that is ‘hard’ for the family of all languages recognized by $k$-lda’s in a way similar to the construction of Sudborough’s hardest languages.

It seems quite natural to seek out a reasonable extension of LOGDCFL by generalizing its underlying machines in a simple way. A basis of LOGDCFL is of course 1dpda’s, each of which is equipped with a read-once input tape together with a storage device called a stack. Each stack allows two major operations. A pop operation is a deletion of a symbol and a push operation is a rewriting of a symbol on the topmost stack cell. However, the usage of pushdown storage seems too restrictive in practice, and thus various extensions of such pushdown automata have been sought in the past literature. For instance, a stack automaton of Ginsburg, Greibach, and Harrison [9, 10] is capable of freely traversing the inside of the stack to access each stored item but it is disallowed to modify them unless the scanning stack head eventually comes to the top of the stack. Thus, each cell of the stack could be accessed
a number of times. Meduna’s deep pushdown automata [19] also allow stack heads to move deeper into the content of the stacks and to replace some symbols by appropriate strings. Other extensions of pushdown automata include [2, 13]. To argue parallel computations, we intend to seek for a reasonable restriction of stack automata by replacing stacks with access-controlled storage devices. Each cell content of such a storage device is modified by its own scanning head, which moves sequentially back and forth along the storage device. This special device and its scanning head naturally allow underlying machines to make more flexible memory manipulations.

In real-life circumstances, it seems reasonable to limit the number of times to access data sets stored in the storage device. For instance, rewriting data items into blocks of a memory device, such as external hard drives or rewritable DVDs, is usually costly and it may be restricted during each execution of a computer program. We thus need to demand that every memory cell on this device can be modified only during the first few accesses and, in case of exceeding the intended access limit, the storage cell turns unusable and no more rewriting is possible. For simplicity, we refer to the number of times that the content of a storage cell is modified as ‘depth’. We need to distinguish two circumstances depending on whether we allow or disallow a free access to ‘new’ input symbols while scanning such unusable data sets. We later use the terms of ‘immunity’ or ‘susceptibility’ to the depth of storage devices. We leave a further discussion on this issue to Section 2.2.

To understand the role of depth limit for an underlying machine, let us consider how to recognize the non-context-free language $L_{abc} = \{a^n b^n c^{2n} | n \geq 0\}$ under an additional requirement that new input symbols are only read while scanning storage cells are not yet frozen. Given an input of the form $a^l b^m c^n$, we first write $a'$ into the first $l$ cells of the storage device, check if $l = m$ by simultaneously reading $b^m$ and traversing the storage device backward by changing $a$ to $b$, and then check if $l + m = n$ by simultaneously reading $c^n$ together with moving the device’s scanning head back and forth by changing $b$ to $c$ and then $c$ to $B$ (frozen blank symbol). This procedure requires its depth limit to be 4.

A storage device whose cells have depth at most $k$ is called a depth-$k$ storage tape in this exposition and its scanning head is hereafter cited as a depth-$k$ storage-tape head for convenience. The machines equipped with those devices, where each storage-tape cell is initially ‘empty’ and turned frozen blank after exceeding its depth limit, are succinctly called one-way deterministic depth-$k$ storage automata (or $k$-sda’s, for short). Our $k$-sda’s naturally expand Hibbard’s $k$-lda’s.2 The requirement of turning cells into frozen blank is imperative because, without it, the machines become as powerful as polynomial-time Turing machines. This statement directly follows from the fact that non-erasing stack automata can recognize the circuit value problem, which is a P-complete problem.

For convenience, we introduce the notation $k$SDA for each index $k \geq 2$ to express the family of all languages recognized by those ‘depth-susceptible’ $k$-sda’s (for a more precise definition, see Section 2.2) whereas the notation $k$SDAimm is reserved for the language family induced by ‘depth-immune’ $k$-sda’s. As the aforementioned example of $L_{abc}$ shows, 4SDA contains even non-context-free languages. Analogously to forming LOGDCFL from DCFL, for any index $k \geq 2$, we define LOG$k$SDA as the closure of $k$SDA under L-m-reductions. It follows from the definitions that LOGDCFL $\subseteq$ LOG$k$SDA $\subseteq$ LOG$(k + 1)$SDA $\subseteq$ P. Among many intriguing questions, we wish to raise the following three simple questions regarding our new language family $k$SDA as well as its L-m-closure LOG$k$SDA.

1. What is the computational complexity of language families $k$SDA as well as LOG$k$SDA?
2. Is there any natural machine model that can precisely characterize LOG$k$SDA in order to avoid the use of L-m-reductions?
3. Is there any language that is L-m-complete for LOG$k$SDA?

The sole purpose of this exposition is to partially answer these questions through Sections 3–5 after a formal introduction of $k$-sda’s in Section 2.
2. Introduction of storage automata

We formally define a new computational model, dubbed as deterministic storage automata, and present basic properties of them.

2.1. Numbers, sets, languages, and Turing machines

We begin with fundamental notions and notation necessary to introduce a new computational model of automata with depth-bounded storage tapes.

The two notations $\mathbb{Z}$ and $\mathbb{N}$ represent the set of all integers and that of all natural numbers (i.e. non-negative integers), respectively. Given two numbers $m, n \in \mathbb{Z}$ with $m \leq n$, $[m, n]_\mathbb{Z}$ denotes the integer interval $\{m, m+1, m+2, \ldots, n\}$. In particular, when $n \geq 1$, we abbreviate $[1, n]_\mathbb{Z}$ as $[n]$. We use the binary representations of natural numbers. For such a representation $x$, the notation $(x)_2$ denotes the corresponding natural number of $x$. For instance, we obtain $(0)_2 = 0$, $(1)_2 = 1$, $(10)_2 = 2$, $(11)_2 = 3$, $(100)_2 = 4$, etc. Given a set $S$, $\mathcal{P}(S)$ denotes the power set of $S$, namely, the set of all subsets of $S$.

An alphabet is a finite nonempty set of ‘symbols’ or ‘letters.’ Given any alphabet $\Sigma$, a string over $\Sigma$ is a finite sequence of symbols in $\Sigma$. The length of a string $x$ is the total number of symbols in $x$ and it is denoted by $|x|$. The special notation $\varepsilon$ is used to express the empty string of length 0. The notation $\Sigma^*$ denotes the set of all strings over $\Sigma$. A language over $\Sigma$ is simply a subset of $\Sigma^*$. As customary, we freely identify a decision problem with its corresponding language. Given a string $x = x_1x_2 \ldots x_n$ with $x_i \in \Sigma$ for all $i \in [n]$, the reverse of $x$ is $x_nx_{n-1} \ldots x_1$ and is denoted $x^R$. For two strings $x$ and $y$ over the same alphabet, $x$ is said to be a prefix of $y$ if there exists a string $z$ for which $y = xz$. In this case, $z$ is called a suffix of $y$. Given a language $A$ over $\Sigma$, $\text{Pref}(A)$ denotes the set of all prefixes of any string in $A$, namely, $\{w \mid \exists y \in \Sigma^*[wy \in A]\}$.

We assume the reader’s familiarity with multi-tape Turing machines and we abbreviate deterministic Turing machines as DTM. To handle a ‘sub-linear’ space DTM, we need to assume that its input tape is read only and an additional rewritable index tape is used to access an input-tape cell whose address is specified by the content of the index tape. More precisely, a machine wants to access the $i$th input-tape cell, then it must write the binary expression of $i$ onto this index tape and then enters a designated ‘query’ state, which triggers to retrieve the content of the $i$th input-tape cell and copy it into the last blank cell of the index tape so that the machine can freely read it. Since the index tape requires $\lceil \log |x| \rceil$ bits to specify each symbol of an input $x$, the space limitation of an underlying machine is thus applied to only work tapes. We assume the reader’s familiarity with fundamental complexity classes, such as CFL, L, NL, and P. Another important complexity class is Steve’s class $\text{SC}^k$, for each index $k \geq 1$, which is the family of all languages recognized by DTM’s in polynomial time using $O(\log^k n)$ space [3]. Let $\text{SC}$ denote the union $\bigcup_{k \in \mathbb{N}^+} \text{SC}^k$. It follows that $L = \text{SC} \subseteq \text{SC}^1 \subseteq \text{SC}^{k+1} \subseteq \text{SC} \subseteq P$.

In order to compute a function on strings, we further provide a DTM with an extra write-once output tape so that the machine produces output strings, where a tape is write-once if its tape head never moves to the left and, whenever its tape head writes a nonempty symbol, it must move to the right. All (total) functions computable by such DTM’s in polynomial time using only logarithmic space form the function class, known as FL.

Given two languages $L_1$ over alphabet $\Sigma_1$ and $L_2$ over $\Sigma_2$, we say that $L_1$ is $L$-m-reducible to $L_2$ (denoted by $L_1 \leq_m^{\Sigma_1} L_2$) if there exists a function $f$ computed by an appropriate polynomial-time DTM using only $O(\log n)$ space such that, for any $x \in \Sigma_1^*$, $x \in L_1$ iff $f(x) \in L_2$. We say that $L_1$ is interreducible to $L_2$ via $L$-m-reductions (denoted by $L_1 \equiv_m^{\Sigma_1} L_2$) if both $L_1 \leq_m^{\Sigma_1} L_2$ and $L_2 \leq_m^{\Sigma_1} L_1$ hold.

We write DCFL for the collection of all deterministic context-free (dcf) languages, which are recognized by one-way deterministic pushdown automata (or 1dpda’s, for short). The notation $\logDCFL$ expresses the $L$-m-closure of DCFL, namely, $\{A \mid \exists B \in \text{DCFL}[A \leq_m^{\Sigma_1} B]\}$.

In relation to our new machine model, introduced in Section 2.2, we briefly explain Hibbard’s ‘scan-limited automata’ [12], which were later reformulated by Pighizzini and Pisoni [20, 21] using a...
single-tape Turing machine model. This exposition follows their formulation. For any positive integer \(k\), a deterministic \(k\)-limited automaton (or a \(k\)-lda, for short) is a single-tape linear automaton with two endmarkers, where an input/work tape initially holds an input string and its tape head can modify the content of each tape cell during the first \(k\) visits (when the tape head makes a turn, we double count the visit) of the tape cell.

2.2. Storage tapes and storage automata

We expand the standard model of pushdown automata by substituting its stack for a more flexible storage device, called a storage tape. Formally, a storage tape is a semi-infinite rewritable tape whose cells are initially blank (filled with a symbol \(\square\)) and are accessed sequentially by a tape head that can move back and forth along the tape by changing tape symbols as it passes through.

In what follows, we fix a constant \(k \in \mathbb{N}^+\). A one-way deterministic depth-\(k\) storage automaton (or a \(k\)-sda, for short) \(M\) is a 2-tape DTM (equipped only with a read-only input tape and a rewritable work tape) of the form \(\langle Q, \Sigma, \{\Gamma^{(e)}\}_{e \in \mathbb{N}^+}, [\{\triangleright, \triangleleft]\}, \delta, q_0, Q_{acc}, Q_{rej}\rangle\) with a finite set \(Q\) of inner states, an input alphabet \(\Sigma\), storage alphabets \(\Gamma^{(e)}\) for indices \(e \in \{0, k\}\) with \(\Gamma = \bigcup_{e \in \{0, k\}} \Gamma^{(e)}\), a transition function \(\delta\) from \((Q - Q_{halt}) \times \tilde{\Sigma} \times \Gamma\) to \(Q \times \Gamma \times D_1 \times D_2\) with \(\tilde{\Sigma} = \Sigma \cup [\triangleright, \triangleleft]\), \(D_1 = \{0, +1\}\), and \(D_2 = \{-1, 0, +1\}\), an initial state \(q_0\) in \(Q\), and sets \(Q_{acc}\) and \(Q_{rej}\) of accepting states and rejecting states, respectively, satisfying both \(Q_{acc} \cup Q_{rej} \subseteq Q\) and \(Q_{acc} \cap Q_{rej} = \emptyset\), provided that \(Q_{halt} = Q_{acc} \cup Q_{rej}\), \(\Gamma^{(0)} = \{\square\}\) (where \(\square\) is a distinguished initial blank symbol), \(\Gamma^{(1)} = \{\triangleright, B\}\) (where \(B\) is a unique frozen blank symbol), and \(\Gamma^{(e_1)} \cap \Gamma^{(e_2)} = \emptyset\) for any distinct pair \(e_1, e_2 \in \{0, k\}\). The sets \(D_1\) and \(D_2\) indicate the directions of the input-tape head and those of the storage-tape head, respectively. The choice of \(D_1\) forces the input tape to be read once. We say that the input tape is read once if its tape head either moves to the right or stays still with scanning no input symbol. A single move (or step) of \(M\) is dictated by \(\delta\). If \(M\) is in inner state \(q\), scanning \(\sigma\) on the input tape and \(\tau\) on the storage tape, a transition \(\delta(q, \sigma, \gamma) = (p, \xi, d_1, d_2)\) forces \(M\) to change \(q\) to \(p\), write \(\xi\) over \(\gamma\), and move the input-tape head in direction \(d_1\) and the storage-tape head in direction \(d_2\). The depth value \(dv(\gamma)\) of a symbol \(\gamma\) is the index \(e\) for which \(\gamma \in \Gamma^{(e)}\).

All tape cells are indexed by natural numbers from left to right, where the least tape cell is the start cell, which is indexed 0. An input tape has the endmarkers \([\triangleright, \triangleleft]\] and a storage tape has only the left endmarker \(\triangleright\). When an input string \(x\) is given to the input tape, it should be surrounded by the two endmarkers as \(\triangleright x \triangleleft\) so that \(\triangleright\) is located at the start cell and \(\triangleleft\) is at the cell indexed \(|x| + 1\). For any index \(i \in \mathbb{N}\), \(x_{(i)}\) denotes the tape symbol written on the \(i\)th input-tape cell, provided that \(x_{(0)} = \triangleright\) (left endmarker) and \(x_{(|x|+1)} = \triangleleft\) (right endmarker). Similarly, when \(z\) represents the non-\(\square\) portion of the content of a storage tape, the notation \(z_{(i)}\) expresses the symbol written in the \(i\)th tape cell. In particular, \(z_{(0)} = \triangleright\).

For the storage tape, we apply the following rewriting rules. Whenever the storage-tape head passes through a tape cell containing a symbol in \(\Gamma^{(e)}\) with \(e < k\), the machine must replace it by another symbol in \(\Gamma^{(e+1)}\), except for the case of the following ‘turns’. Remember that no symbol in \(\Gamma^{(k)}\) can be modified at any time. We distinguish two types of turns. Although we use various storage alphabets \(\Gamma^{(0)}, \Gamma^{(1)}, \ldots, \Gamma^{(k)}\), since those alphabets are mutually disjoint, we can easily discern from which direction the tape head arrives simply by reading a storage tape symbol written in each tape cell. As customary, we explicitly demand that, on a storage tape, no machine writes \(\triangleright\) over any non-\(\triangleright\) symbol. A left turn at step \(t\) refers to \(M\)'s step at which, after \(M\)'s tape head moves from the left at step \(t-1\), it moves to the left at step \(t\). In other words, \(M\) takes two transitions \(\delta(q, \sigma_1, \gamma_1) = (p, \tau, d_1, +1)\) at step \(t-1\) and \(\delta(p, \sigma_2, \gamma_2) = (r, \xi, e, -1)\) at step \(t\). Similarly, we say that \(M\) makes a right turn at step \(t\) if \(M\)'s tape head moves from the right at step \(t-1\) and changes its direction to the right at step \(t\). Whenever a tape head makes a turn, we treat such a case as ‘double accesses.’ More formally, at a turn, any symbol in \(\Gamma^{(e)}\) with \(e < k\) must be changed to another symbol in \(\Gamma^{(\min\{k, e+2\})}\).
The depth-\( k \) requirement demands that, assuming that \( M \) modifies \( \sigma \in \Gamma^{(c)} \) written on a storage tape with \( e \in [0, k - 1]_{\geq} \) to \( \tau \) and moves its storage-tape head in direction \( d \), if \( d = (-1)^{e} \) (i.e. making no turn), then \( \tau \) must belong to \( \Gamma^{(\min(e+1),d)} \), and, if \( d = (-1)^{e+1} \) (i.e. making a turn), then \( \tau \) must be in \( \Gamma^{(\min(e+2,k))} \). A storage tape that satisfies the depth-\( k \) requirement is succinctly called a depth-\( k \) storage tape.

Instead of making \( \varepsilon \)-moves (i.e. a tape head neither moves nor reads any tape symbol), we allow a tape head to make a stationary move,\(^3\) by which the tape head stays still and the currently scanned symbol is unaltered. The tape head direction ‘0’ indicates such a stationary move. For clarity, if the input-tape (resp., the storage-tape) head makes a stationary move, then we call this move an input-stationary move (resp., a storage-stationary move).

Let us consider two different models whose input-tape head is either ‘depth-susceptible’ or ‘depth-immune’ to the content of each storage-tape cell. Recall the blank-skipping property of a \( k \)-lda [26], which in essence states that, while the \( k \)-lda is scanning the blank symbol, it should not change its inner state. The \( k \)-sda \( M \) is called depth-susceptible if, for any storage symbol \( \gamma \) scanned currently, (i) if \( \gamma \) is in \( \Gamma^{(k-1)} \cup \Gamma^{(k)} \), then the input-tape head must make a stationary move, and (ii) if \( \gamma \) is frozen blank, then the current inner state should not change; namely, for any transition \( \delta(q,\sigma,\gamma) = (p,\xi,d_{1},d_{2}) \), (i’) \( \gamma \in \Gamma^{(k-1)} \cup \Gamma^{(k)} \) implies \( d_{1} = 0 \) and (ii’) \( \gamma = B \) implies \( q = p \). On the contrary, the machine is depth-immune if there is no restriction.

A surface configuration of \( M \) on input \( x \) is of the form \( (q, l_{1}, l_{2}, z) \) with \( q \in Q \), \( l_{1} \in [0,|x| + 1]_{\geq} \), \( l_{2} \in \mathbb{N} \), and \( z \in (\Gamma - \{\square\})^{*} \), which indicates the situation where \( M \) is in inner state \( q \), the storage tape contains \( z \) (except for the tape symbol \( \square \)), and two tape heads scan the \( l_{1} \)th cell of the input tape and the \( l_{2} \)th cell of the storage tape.

The initial surface configuration has the form \( (q_{0},0,0,\succ) \) and \( \delta \) describes how to reach the next surface configuration in a single step. For convenience, we define the depth value \( dv(C) \) of a surface configuration \( C = (q, l_{1}, l_{2}, z) \) to be the depth value \( dv(\varepsilon_{(l_{2})}) \). An accepting surface configuration (resp., a rejecting surface configuration) is of the form \( (q, h_{1}, h_{2}, w) \) with \( q \in Q_{ac} \) (resp., \( q \in Q_{rej} \)). A halting configuration means either an accepting configuration or a rejecting configuration. A computation of \( M \) on input \( x \) starts with the initial surface configuration with the input \( x \) and either ends with a halting surface configuration or continues forever. The \( k \)-sda \( M \) accepts (resp., rejects) \( x \) if the computation of \( M \) on \( x \) reaches an accepting surface configuration (resp., a rejecting surface configuration). For readability, we will drop the word ‘surface’ altogether in the subsequent sections since we use only surface configurations.

For a language \( L \) over \( \Sigma \), we say that \( M \) recognizes (accepts or solves) \( L \) if, for any input string \( x \in \Sigma^{*} \), (i) if \( x \in L \), then \( M \) accepts \( x \) and (ii) if \( x \notin L \), then \( M \) rejects \( x \). This implies that \( M \) halts within finite steps on all inputs. In this case, \( L \) is also expressed as \( L(M) \). For two \( k \)-sda’s \( M_{1} \) and \( M_{2} \) over the same input alphabet \( \Sigma \), we say that \( M_{1} \) is (computationally) equivalent to \( M_{2} \) if, for any input \( x \in \Sigma^{*} \), \( M_{1} \) accepts (resp., rejects) \( x \) iff \( M_{2} \) accepts (resp., rejects) \( x \).

For notational convenience, we write \( k \text{SDA} \) for the collection of all languages recognized by depth-susceptible \( k \)-sda’s and \( \log k \text{SDA} \) for the collection of languages that are \( L \)-m-reducible to certain languages in \( k \text{SDA} \). Moreover, we set \( \omega \text{SDA} \) to be the union \( \bigcup_{k \in \mathbb{N}^{+}} k \text{SDA} \). With this notation, the non-context-free language \( L_{abc} \), discussed in Section 1, belongs to \( 4 \text{SDA} \). Thus, \( 4 \text{SDA} \not\subseteq \text{CFL} \) follows instantly. Based on the depth-immune model of \( k \)-sda, we similarly define \( k \text{SDA}_{imm} \), \( \omega \text{SDA}_{imm} \), and \( \log k \text{SDA}_{imm} \).

The depth-susceptibility plays a crucial role in showing the following equality between \( 2 \text{SDA} \) and \( \text{DCFL} \). This fact supports the claim that \( k \)-sda’s naturally expand 1dpda’s. It follows that \( \text{DCFL} \subseteq k \text{SDA} \subseteq (k + 1) \text{SDA} \subseteq \omega \text{SDA} \) for any \( k \geq 2 \).

**Lemma 2.1:** \( 2 \text{SDA} = \text{DCFL} \).

**Proof:** It was shown that \( \text{DCFL} \) is precisely characterized by deterministic 2-limited automata (or 2-lda’s, for short) [12, 21]. Since any 2-lda can be transformed to another 2-lda with the blank-skipping
property [25], depth-susceptible 2-sda’s can simulate 2-lda’s; thus, we immediately conclude that DCFL ⊆ 2SDA.

For the converse, it suffices to simulate depth-susceptible 2-sda’s by appropriate one-way deterministic pushdown automata (or 1dpda’s). Given a depth-susceptible 2-sda M, we design a 1dpda N that works as follows. We want to treat the storage-tape of M as a stack by ignoring the frozen blank symbol B in Γ(2). When M modifies the initial blank symbol □ on its storage tape to a new symbol σ in Γ(1), N pushes σ to the stack. In the case where M modifies a storage symbol σ to B, N pops the same symbol σ from the stack. Note that, since M is depth-susceptible, it cannot move the input-tape head. While M scans B, N does nothing. As for the behaviour of N’s input-tape head, if M reads an input symbol σ and moves to the right, then N does the same. On a tape containing σ ∈ Γ(1), since M’s input-tape head makes a series of stationary moves, N reads σ at the first move, remembers σ, and makes ε-moves afterwards until M ends its stationary moves. Obviously, the resulting machine is a 1dpda and it precisely simulates M.

Remark: Remember that storage-tape cells of k-sda’s become blank after the first k accesses. If we allow such tape cells to freeze the last written symbols and preserve them forever, instead of erasing them, then the resulting machines get enough power to recognize even the circuit value problem, which is P-complete under L-m-reductions. In this exposition, we do not further delve into this topic.

3. Two machine models that characterize LOGkSDA

We begin with a study on the structural properties of LOGkSDA, which is the closure of kSDA under L-m-reductions. In particular, we intend to seek out different characterizations of LOGkSDA with no use of L-m-reductions. The basic idea of the elimination of such reductions is attributed to Sudborough [23], who characterized LOGDCFL using two machine models: polynomial-time log-space auxiliary deterministic pushdown automata and polynomial-time multi-head deterministic pushdown automata. Our goal in this section is to expand these machine models to fit into the framework of depth-k storage automata and prove their characterizations of LOGkSDA.

3.1. Deterministic auxiliary depth-k storage automata

We expand deterministic auxiliary pushdown automata to deterministic auxiliary depth-k storage automata, each of which is equipped with a two-way read-only input tape, a rewritable auxiliary (work) tape, and a depth-k storage tape.

Let us formally formulate deterministic auxiliary depth-k storage automata (or an aux-k-sda, for short). For the description of such a machine, firstly we prepare a two-way read-only input tape and a depth-k storage tape and secondly we supply a new space-bounded rewritable auxiliary tape whose cells are freely modified by a two-way tape head. An aux-k-sda M is a 3-tape DTM (Q, Σ, Θ, {Γ(e)}e∈[0,k]Z, {>, <}, δ, q0, Qacc, Qrej) with a read-only input tape, an auxiliary rewritable (work) tape with an alphabet Θ, and a depth-k storage tape. Initially, the input tape is filled with > x <, the auxiliary tape is blank, and the depth-k storage tape has only the initial blank symbols □ except for the left endmarker >. We set Γ(0) = {□}, Γ(k) = {>, B}, and Γ = ∪e∈[0,k]Z Γ(e), provided that Γ(e1) ∩ Γ(e2) = ∅ for any distinct pair e1, e2 ∈ [0, k]Z. The transition function δ of M maps (Q − Qhalt) × Σ × Θ × Γ to Q × Θ × Γ × D1 × D2 × D3, where Qhalt = Qacc ∪ Qrej, Σ = Σ ∪ {>, <}, and D1 = D2 = D3 = {−1, 0, +1}. A transition δ(q, σ, τ, γ) = (p, θ, ξ, d1, d2, d3) indicates that, on reading input symbol σ, M changes its inner state q to p by moving an input-tape head in direction d1, changes auxiliary tape symbol τ to θ by moving an auxiliary-tape head in direction d2, and changes storage tape symbol γ to ξ by moving in direction d3. A string x is accepted (resp., rejected) if M enters an inner state in Qacc (resp., Qrej).

When excluding (Θ, D2) from the definition of M, the resulting automaton must fulfill the depth-k requirement of k-sda’s given in Section 2.2. The depth-susceptibility condition for an aux-k-sda is
stated as follows: For a transition \( \delta(q, \sigma, \tau, \gamma) = (p, \theta, \xi, d_1, d_2, d_3) \), (i) if \( \gamma \in \Gamma^{(k-1)} \cup \Gamma^{(k)} \), then \( d_1 = d_2 = 0 \) and (ii) if \( \gamma = B \), then \( q = p \). Regarding stationary moves, \( M \) should satisfy the stationary requirement that, assuming that \( \delta(q, \sigma, \tau, \gamma) = (p, \theta, \xi, d_1, d_2, d_3) \), \( d_2 = 0 \) implies \( \tau = \theta \) and \( d_3 = 0 \) implies \( \gamma = \xi \).

Let us assume that, for a certain positive integer \( c \), \( M \)'s auxiliary tape uses at most \( c \lceil \log n \rceil \) tape cells on any input of length \( n \geq 2 \). It is then possible to reduce this space bound down to \( \lceil \log n \rceil \). We first introduce a larger auxiliary tape alphabet \( \Theta' \) so that the content of \( c \) consecutive tape cells can be expressed as a single symbol in \( \Theta' \). Since a tape head cannot access all \( c \) tape cells at once, we need to specify which tape cell is currently scanned by the tape head. For this purpose, we use a number \( i \in [c] \) to refer to the \( i \)th tape cell and make an inner state hold the information on \( i \). Therefore, without loss of generality, we can assume that \( M \) uses at most \( \lceil \log n \rceil \) tape cells on the auxiliary tape.

### 3.2. Multi-head deterministic depth-\( k \) storage automata

We further introduce another useful machine model by expanding two-way multi-head pushdown automata to two-way multi-head deterministic depth-\( k \) storage automata, each of which allows multiple two-way tape heads to read a given input.

For each fixed integer \( \ell \geq 1 \), we define an \( \ell \)-head deterministic depth-\( k \) storage automaton as a 2-tape DTM with \( \ell \) read-only depth-susceptible tape heads scanning over an input tape and another tape head over a depth-\( k \) storage tape. For convenience, we call such a machine a \( k\text{-sda}_2(\ell) \), where the subscript ‘2’ emphasizes that all tape heads move in both directions (including stationary moves). Remember that each \( k\text{-sda}_2(\ell) \) has actually \( \ell + 1 \) tape heads, including one tape head working along the storage tape.

More formally, a \( k\text{-sda}_2(\ell) \) is a tuple \((Q, \Sigma, \{\Gamma^{(e)}\}_{e \in [0,k]Z}, [\sigma, \tau], \delta, q_0, Q_{\text{acc}}, Q_{\text{rej}}) \) with a transition function \( \delta \) mapping \((Q - Q_{\text{halt}}) \times \tilde{\Sigma} \times \Gamma \) to \( Q \times \Gamma \times D^\ell \times D \), where \( Q_{\text{halt}} = Q_{\text{acc}} \cup Q_{\text{rej}} \), \( \tilde{\Sigma} = \Sigma \cup [\sigma, \tau] \), \( D = \{-1, 0, +1\} \), \( \Gamma^{(0)} = \{\Box\} \), \( \Gamma^{(k)} = [\sigma, B] \), and \( \Gamma = \bigcup_{e \in [0,k]Z} \Gamma^{(e)} \), provided that \( \Gamma^{(e_1)} \cap \Gamma^{(e_2)} = \emptyset \) for any distinct pair \( e_1, e_2 \in [0,k]Z \). A transition of the form \( \delta(q, \sigma_1, \ldots, \sigma_\ell, \gamma) = (p, \xi, d_1, \ldots, d_\ell, d_{\ell+1}) \) means that, if \( M \) is in inner state \( q \), scanning a tuple \((\sigma_1, \ldots, \sigma_\ell)\) of symbols on the input tape by the \( \ell \) read-only tape heads as well as symbol \( \gamma \) on the depth-\( k \) storage tape by the rewritable tape head, then, in a single step, \( M \) enters inner state \( p \) and writes \( \xi \) over \( \gamma \) by moving the \( \ell \)th input-tape head in direction \( d_\ell \) for every index \( \ell \in [\ell] \) and the storage-tape head in direction \( d_{\ell+1} \). Note that a \( k\text{-sda}_2(1) \) and a \( k\text{-sda} \) are similar in their machine structures but the former can move its input-tape head to the left.

The acceptance/rejection criteria are the same as those of underlying \( k\text{-sda} \)’s. We demand that any \( k\text{-sda}_2(\ell) \) should satisfy the depth-susceptibility condition, which asserts that, for any transition \( \delta(q, \sigma_1, \sigma_2, \ldots, \sigma_\ell, \gamma) = (p, \xi, d_1, d_2, \ldots, d_\ell, d_{\ell+1}) \) of \( M \), (i) \( \gamma \in \Gamma^{(k-1)} \cup \Gamma^{(k)} \) implies \( d_1 = d_2 = \cdots = d_\ell = 0 \) and (ii) \( \gamma = B \) implies \( q = p \).

### 3.3. Characterization theorem

We intend to demonstrate that the two new machine models introduced in Sections 3.1–3.2 precisely characterize LOGkSDA. This result can be seen as a natural extension of Sudborough’s machine characterization of LOGDCFL to LOGkSDA.

**Theorem 3.1:** Let \( k \geq 2 \). Let \( L \) be any language. The following three statements are logically equivalent.

1. \( L \) is in LOGASDA.
2. There exists an aux-\( k\text{-sda} \) that recognizes \( L \) in polynomial time using logarithmic space.
3. There exist a number \( \ell \geq 2 \) and a \( k\text{-sda}_2(\ell) \) that recognizes \( L \) in polynomial time.

In the rest of this subsection, we intend to prove Theorem 3.1. Firstly, Sudborough’s proof [23, Lemmas 3–6] for LOGDCFL relies on the heavy use of stack operations, which are applied only to
the topmost symbol of the stack but the other symbols in the stack are intact. In our case, however, we need to deal with the operations of a storage-tape head, which can move back and forth along a storage tape by modifying cell’s content as many as \( k \) times. Secondly, Sudborough’s characterization utilizes a simulation procedure [11, pp.338–339] of Hartmanis and a proof argument [7, Lemma 4.3] of Galil; however, we cannot directly use them, and thus a new idea is needed to establish Theorem 3.1. The proof of this theorem therefore requires technically challenging simulations among \( FL \)-functions and the other machine models of aux-\( k \)-sda and \( k \)-sda\(_2(\ell)\).

**Lemma 3.2:** Given a function \( f : \Sigma_1^* \rightarrow \Sigma_2^* \) in \( FL \) for certain alphabets \( \Sigma_1 \) and \( \Sigma_2 \) and a depth-susceptible \( k \)-sda \( M \) working over \( \Sigma_2 \) in polynomial time, there exists a log-space aux-\( k \)-sda \( N \) that recognizes \( L = \{ x \in \Sigma_1^* \mid f(x) \in L(M) \} \) in polynomial time.

**Proof:** Let \( f : \Sigma_1^* \rightarrow \Sigma_2^* \) be any function in \( FL \). We take a DTM \( M_f \), equipped with a read-only input tape, a logarithmic space-bounded rewritable work tape, and a write-once output tape, and assume that \( M_f \) computes \( f \) in polynomial time. A given depth-susceptible \( k \)-sda running in polynomial time is denoted \( M \). We then set \( L \) to be the language \( \{ x \mid f(x) \in L(M) \} \).

We design the desired aux-\( k \)-sda \( N \) for \( L \) as follows. Given any input \( x \) in \( \Sigma_1^* \), we repeat the following process until \( M \) enters a certain halting state. Using an auxiliary tape of \( N \), we keep track of the content on \( M_f \)’s work tape, two head positions of \( M \)'s input and storage tapes and \( M_f \)’s input tape. Since \( N \) shares the same input \( x \) with \( M, N \) can precisely simulate the movement of \( M \)’s input-tape head. We intend to force \( N \) to scan the same storage symbol as \( M \). Let \( \gamma \) denote the storage symbol scanned currently by \( M \). Assume that \( M \)'s input-tape head is located at cell \( h \in [0, |f(x)| + 1]_Z \).

1. If \( M \) makes an input-stationary move, then we simply simulate one step of the behaviour of \( M \)'s storage-tape head since we can reuse the last produced output symbol of \( M_f \). We thus move the storage-tape head of \( N \) in the same direction as \( M \).
2. Assume that \( M \) moves its input-tape head from the left to cell \( h \). We remember the current location of \( M \)'s input-tape head, return \( N \)'s input-tape head to the last location of \( M_f \)'s input-tape head, and resume the simulation of \( M_f \) with the use of \( N \)'s auxiliary tape as a work tape by making a (possible) series of storage-stationary moves of \( N \) until \( M_f \) produces the \( h \)th output symbol, say, \( \sigma \). This is possible because the output tape of \( M_f \) is write-once, and thus there is no need to recompute any past output symbols. Once \( \sigma \) is obtained, \( N \) remembers the positions of \( M_f \)'s tape heads, moves its input-tape head back to the previous location. We next simulate a single step of \( M \) on \( \sigma \) together with the storage symbol. We then update the tape-head positions.

It is not difficult to show that \( N \) eventually reaches the same type (accepting or rejecting) of halting states as \( M \) does within polynomially many steps. Notice that the storage-tape head and the auxiliary-tape head do not work simultaneously. The depth-susceptibility of \( N \) comes from that of \( M \) since \( M_f \) is simulated only when \( M \)'s storage-tape head reads a symbol not in \( \Gamma^{(k-1)} \cup \Gamma^{(k)} \). Thus, \( N \) is indeed an aux-\( k \)-sda.

Given an aux-\( k \)-sda \( M \), we intend to construct a \( k \)-sda\(_2(\ell)\), which mimics the behaviour of a given aux-\( k \)-sda, where \( \ell \) is an appropriate constant depending only on \( M \).

**Lemma 3.3:** Let \( k \geq 2 \). Let \( M \) denote a polynomial-time log-space aux-\( k \)-sda, there are a constant \( c > 0 \) and a \( k \)-sda\(_2(5c + 2) \) \( N \) that simulates \( M \) in polynomial time.

**Proof:** Let \( k \geq 2 \) and let \( M = (Q, \Sigma, \{ \Gamma^{(e)} \}_{e \in [0,k]}_Z, \{ \sigma, \delta \}, q_0, Q_{acc}, Q_{rej}) \) denote any aux-\( k \)-sda that runs in polynomial time using logarithmic space on all inputs of length \( n \geq 2 \).

As noted in Section 3.1, we can assume that \( M \) uses at most \( |\log n| \) auxiliary-tape cells, where \( n \) indicates input length. Let \( c = \lceil \log |\Theta| \rceil \) and assume that \( \Theta = \{ \theta_1, \theta_2, \ldots, \theta_{2^c} \} \). We identify each
element \( \theta_i \) in \( \Theta \) with \( i \), which is further expressed as its binary string of length \( c \). We partition every auxiliary tape cell into \( c \) blocks, indexed by 1, 2, \ldots, \( c \). A symbol \( \theta_i \) can be stored in those blocks in such a way that, for any \( j \in [c] \), the \( j \)th bit of \( \theta_i \) is placed in the \( j \)th block. By fixing each block index for all tape cells, we can partition an entire tape into \( c \) separate ‘tapes’, which are customarily called tracks.

We want to construct a polynomial-time \( k \)-sda\(_2(5c + 2) \) \( N \) for which \( L(N) \) coincides with \( L(M) \). Other than a storage-tape head, we use \( M \)’s input-tape head as the principal tape head of \( N \). We further introduce additional \( 5c \) tape heads to simulate the behaviour of an auxiliary-tape head of \( M \).

In what follows, we fix one of the \( c \) tracks of the auxiliary tape. If each track contains a string \( w \) of length \( \lceil \log N \rceil \), we treat it as the binary number \( (1w^c)^{(2)} \). We use two tape heads to remember the positions of the input-tape head and the auxiliary-tape head of \( M \). To remember the number \( (1w^k)^{(2)} \), we need additional 5 tape heads (other than the storage-tape head).

Head 1 keeps the tape head position. Whenever \( M \) moves the auxiliary-tape head, \( N \) moves head 1 as well. Head 2 moves backward to measure the distance \( l \) of the auxiliary-tape head from the left end of the auxiliary tape. Using this information, \( N \) moves head 3 as follows. If the tape head changes 0 to 1 (resp., 1 to 0) on this target track, then \( N \) moves the head \( 2^l \) cells to the right (resp., to the left). How can we move a tape head to cell \( 2^l \) from \( \succ \)? Following [11], we use three tape heads to achieve this goal in the following way. We move head 4 one cell to the right. As head 4 takes one step on the way back to \( \succ \), we move head 5 two cells to the right. We then switch the roles of heads 4 and 5. As head 5 takes one step back toward \( \succ \), we move head 4 two cells to the right. If we repeat this process \( t \) times, one of the heads indeed reaches cell \( 2^l \). Hence, for the \( t \)th run, head 3 reaches cell \( 2^l \). This process requires 3 tapes. Thus, the total of 5 tape heads are sufficient to simulate the operation on the track content of the auxiliary tape.

Since there are \( c \) tracks, we need the total of \( 5c + 2 \) heads (including the input-tape head and the storage-tape head) for our intended simulation. Note that, while the additional \( 5c \) tape heads are moving, the storage-tape head scans no symbol in \( \Gamma^{(k−1)} \cup \Gamma^{(k)} \).

We lessen the number of input-tape heads from \( \ell + 2 \) to \( \ell + 1 \) for any \( \ell \geq 1 \) by implementing a ‘counter head’ to measure how far the tape head is away from a particular tape cell. A counter head is simply a two-way depth-susceptible tape head moving on an input tape in such a way that, once this tape head is activated, it starts moving from \( \succ \) to the right, stays still for a while, and comes back to \( \succ \) with non-stopping movement (except for the period of reaching the rightmost location) by ignoring all input symbols on its way. We also require the storage-tape head to stay still during the activation of this counter head.

We partition every tape cell into two blocks, called the upper track and the lower track. The content of each tape cell is expressed by the track notation as \( [\tau_\ell \sigma_\ell] \) if two tape symbols \( \sigma \) and \( \tau \) are written in those tracks [24].

**Lemma 3.4:** Let \( k \geq 2 \) and \( \ell \geq 1 \). Given any \( k \)-sda\(_2(\ell + 2) \) \( M \) with a counter head over input alphabet \( \Sigma \) running in polynomial time, there exists a polynomial-time \( k \)-sda\(_2(\ell + 1) \) with a counter head that recognizes the language \( L_{ab} = \{ \tilde{x} \mid x \in L(M) \} \), where \( a \) and \( b \) are distinct tape symbols not in \( \Sigma \), where \( \tilde{x} = (ax_1b)(ax_2b) \cdots (ax_nb) \) with \( \tilde{x}_i = [\tau_{\ell_i}] [\sigma_{\ell_i}] \cdots [\sigma_{\ell_i}] \) for \( x = x_1x_2 \cdots x_n \).

**Proof:** We fix \( k \geq 2 \) and \( \ell \geq 1 \). Let \( M \) denote any polynomial-time \( k \)-sda\(_2(\ell + 2) \) with a counter head. Among all read-only tape heads indexed from 1 to \( \ell \), we call head 1 the principal-tape head and call head 2 and head 3 subordinate tape heads. We wish to simulate these three tape heads by two tape heads, eliminating one subordinate tape head. We leave all the remaining tape heads ‘unmodified’ in the sense that they work only over \( \tilde{x}_1 \) in \( ax_1b \) similarly to the original tape heads working over \( x \). This simulation can be carried out on an appropriate polynomial-time \( k \)-sda\(_2(\ell + 1) \), say, \( N \). Including \( a \) and \( b \), we set \( t = |axb| \). The associated input to \( N \) is \( \succ \tilde{x} \). Initially, heads 1–3 of \( M \) are all stationed at cell 0.
We intend to simulate each step of $M$ by conducting a series of steps of $N$. Assume that head 2 is currently located at cell $i$ and head 3 is at cell $j$. Such a location pair is expressed as $(i, j)$. For any index $i \in [1]$, we call the block $a^i b$ of the input $\bar{x}$ as block $i$. For convenience, $\triangleright$ and $\triangleleft$ are respectively called block 0 and block $n + 1$. We try to ‘express’ the pair $(i, j)$ by stationing a designated tape head of $N$, say, head $h$, at the $i$th symbol of the $j$th block of $N$’s input tape. Hereafter, we assume that head $h$ is currently located at the $i$th symbol of the $j$th block. We force $N$ to remember two input symbols, say, $(\sigma_i, \sigma_j)$ written at the $i$th and the $j$th cells of $M$’s input tape.

We further assume that, in a single step, heads 2 and 3 of $M$ move from $(i, j)$ to a new location pair $(i + d_1, j + d_2)$ by the given head directions $d_1, d_2 \in \{-1, 0, +1\}$. To simulate this single step of $M, N$ needs to move head $h$ to a new location and read two symbols $(\sigma_{i+d_1}, \sigma_{j+d_2})$. If $d_2 = 0$, then $N$ moves head $h$ in direction $d_1$. Otherwise, $N$ moves head $h$ in direction $d_1$, reads its input symbol $\sigma_{i+d_1}$, and remember it using the inner state. Next, $N$ moves head $h$ leftward to the nearest $a$. As the head moves, $N$ also moves the counter head to the right (from the start cell) to count the number $i + d_1$. Furthermore, $N$ moves head $h$ to the symbol $a$ in block $j + d_2$ without moving the counter head. We restart the counter head backwardly. Finally, $N$ moves head $h$ rightward together with the counter head until the counter head comes back to the start cell. At the time when the counter head arrives at the start cell, head $h$ reaches the $(i + d_1)$th cell of block $j + d_2$. Head $h$ then reads the input symbol $\sigma_{j+d_2}$ and $N$ remembers $(\sigma_{i+d_1}, \sigma_{j+d_2})$ using its inner state. Note that the tape head on the storage tape never moves during the above process.

After the simulation of $M, N$ clearly reaches the same type (accepting or rejecting) of a halting state as $M$ does.

Notice that a $k$-sda owns a one-way input-tape head whereas a $k$-sda$_2(2)$ uses a two-way input-tape head. In the next lemma, we need to simulate two-way head moves of a $k$-sda$_2(2)$ using one-way head moves of another $k$-sda$_2(2)$. For this purpose, we utilize a ‘counter’ again together with the use of the reverse of an input.

**Lemma 3.5:** Given a polynomial-time $k$-sda$_2(2)$ $M$ with a counter head, there exists another $k$-sda$_2(2)$ $N$ with a counter head such that (i) $N$’s input-tape head never moves to the left and (ii) $N$ recognizes the language $L_{rev} = \{ \bar{x} \mid x \in L(M) \}$ in polynomial time, where $\bar{x} = (ax#x^Rb)^{|axb|}$ and $#$ denotes a new separator.

**Proof:** Let $M$ be any polynomial-time $k$-sda$_2(2)$ with a counter head. We simulate the two-way movement of $M$’s input-tape head by a one-way tape head in the following way. Assume that $i$ represents the cell position of $M$’s input-tape head. Let $\sigma_i$ and $\sigma_{i-1}$ respectively denote the tape symbols at cells $i$ and $i−1$. We briefly call by head 1 $M$’s input-tape head other than the counter head. If head 1 moves to the right or makes a stationary move, then $N$ simulates the step of $M$ exactly. In what follows, we consider the case where head 1 moves to the left; that is, the new position of head 1 is $i−1$. By the depth-susceptibility condition of $M$, the current storage-tape cell contains no symbol in $\Gamma^{(k−1) \cup \Gamma^{(k)}}$ because, otherwise, head 1 cannot move. In this case, we move $N$’s input-tape head and the counter head simultaneously to the right until the tape head reaches the first encounter of $b$ in $ax#x^Rb$. We then continue moving the tape head rightward while we move the counter head back to the left endmarker. We finally make the tape head shift to the right cell to reach the symbol $\sigma_{i−1}$ in $ax#$.

Next, we show how to eliminate a counter head using the fact that the counter head is depth-susceptible.

**Lemma 3.6:** Let $M$ denote any polynomial-time $k$-sda$_2(2)$ $M$ with a one-way input-tape head and a counter head. There exists a depth-susceptible $k$-sda $N$ that recognizes $L’ = \{ \bar{x} \mid x \in L(M) \}$ in polynomial time, where $\bar{x} = x_11^n x_21^n \ldots 1^n x_n$ for $x = x_1 x_2 \ldots x_n$. 
Sudborough made a similar claim, whose proof relies on Galil’s argument [7], which uses a stack to store and remove specific symbols in order to remember the distance of a tape-head location from a particular input tape cell. However, since tape cells on the storage tape are not allowed to modify more than $k$ times, we need to develop a different strategy to prove Lemma 3.6. For this purpose, we run an additional procedure of making enough open space on the depth-$k$ storage tape for future recording of the tape-head’s movement.

**Proof:** Take any polynomial-time $k$-sda$_2(2)$ $M$ of the form $(Q, \Sigma, \Gamma, \delta, q_0, Q_{acc}, Q_{reg})$ with a counter head. We wish to simulate the behaviour of the counter head using a storage tape of the desired $k$-sda $N$. In what follows, we describe this simulation procedure. Let $x$ denote any input and set $n = |\triangleright x \triangleleft|$. To simplify the description of the intended simulation by a $k$-sda $N$, we deliberately allow $N$ to change any symbol not in $\Gamma^{(k)}$ into $B$ in a single step.

Recall that, whenever the counter head is activated, it starts at $\triangleright$, moves rightward for a certain number of steps, say, $i$, and moves back to the start cell to complete a single round of the ‘counting’ process. During this process, the storage-tape head stays still. Since we force $N$ to move its input-tape exactly in the same direction as $M$’s, in what follows, we focus only on the simulation of $M$’s counter and storage-tape heads. Using its inner states, $N$ can remember (a) which direction the storage-tape head comes from and (b) the contents of the tape cell currently scanned and its left and right adjacent tape cells (if any).

For convenience, we set $\gamma^{(j,k)} = \gamma^{(0,k)} = B$ and $B^{(k)} = \hat{B}^{(k)} = B$. We introduce special storage alphabets $\Gamma^{(j)}_N (j \in [0,k]_\mathbb{Z})$ for $N$. Firstly, we set $\Gamma^{(0)}_N = \Gamma^{(0)}$ and $\Gamma^{(k)}_N = \Gamma^{(k)}$. For any $j \in [k-1]$, the storage alphabet $\Gamma^{(j)}_N$ in part consists of symbols of the form $\gamma^{(0,e)}$ and $\gamma^{(e,0)}$ for $e \in [0,k]_\mathbb{Z}$ associated with $\gamma$ in $\Gamma^{(j)}$. Moreover, $\Gamma^{(j)}_N$ contains symbols $B^{(j)}$ and $\hat{B}^{(j)}$.

We partition the storage tape into a number of ‘regions’, where a region consists of 2$k$ blocks, each of which contains $n$ tape cells. Each region is meant to simulate one run of the counter head and it basically holds the information on one storage symbol. Two regions are separated by one special ‘separator block’ of $n$ cells. See an illustration in Figure 1 for blocks and regions. Each block contains a string, which has one of the following three forms: $\gamma' B_1 B_2 \ldots B_{n-1}$ with $\gamma' \in \Gamma_N - \{\square, \triangleright\}$, $B_1 B_2 \ldots B_{n-1} \gamma'$ with $\gamma' \in \Gamma_N - \{\triangleright\}$, and $B_1 B_2 \ldots B_n$, where $B_i \in \{B^{(c)}, \hat{B}^{(c)} \mid c \in [0, k-1]_\mathbb{Z}\} \cup \{B\}$ for any $i \in [n]$. If $\gamma' \neq B$, then $\gamma'$ is called a representative of a block. A tape cell in a block is also called a representative if it contains a representative of the block. A block is called active if it contains a representative, and all other blocks are called passive. In particular, we call a block consumed if it is filled with $B$, and thus there is no representative. We say that a block is blank if the block consists only of $B$. The parameter $e$ in $\gamma^{(0,e)}$ (resp., $\gamma^{(e,0)}$) indicates the existence of $e$ consumed blocks in the area of the region that are left (resp., right) to the currently scanned cell.

![Figure 1](image-url)

**Figure 1.** A simulation of the movement of the storage-tape head and the counter head of $M$. (1) After moving from $\gamma_1$ to $\gamma_2$, an input-tape-head of $M$ changes $\gamma_2$ to $\gamma_{new}$ and moves in direction $d = +1$. (2) The counting head starts at cell 0, travels through $i$ cells, and returns to cell 0. (3) The storage-tape head of $N$ moves as depicted in this figure to simulate (1) and (2).
In a run of the procedure described below, we maintain the circumstances, in which there is at most one active block in each region. Assume that \( \gamma_1, \gamma_2, \gamma_3 \) is the content of three neighboring cells, the middle of which is being scanned by \( M \)'s storage-tape head, provided that, whenever \( \gamma_2 \) equals \( \triangleright \), we automatically ignore \( \gamma_1 \). Assuming that \( \gamma'_1, \gamma'_2, \) and \( \gamma'_3 \) are three representatives associated respectively with \( \gamma_1, \gamma_2, \) and \( \gamma_3 \). Assume that \( M \)'s tape head stays over \( \gamma_2 \) and \( N \)'s tape head does over \( \gamma'_2 \).

Let us focus on a computation of \( M \) on input \( x \) and simulate this computation on the \( k \)-sda \( N \). To make this simulation easier, we partition the computation into a series of 'session steps,' each of which constitutes a consecutive sequence of \( M \)'s moves defined as follows. Assume that, at time \( t \), the storage-tape head of \( M \) is resting on a cell containing a non-\( B \) symbol, say, \( \gamma_0 \). We consider a consecutive cells holding a string of the form \( \gamma_{-t_1} \gamma_{-t_1+1} \ldots \gamma_{-1}\gamma_0 \gamma_1 \ldots \gamma_{t_2-1}\gamma_{t_2} \) satisfying that \( t_1, t_2 \in \mathbb{N}, \gamma_{-t_1} \notin \{\square, B\}, \gamma_{t_2} \neq B, \) and \( B_i = B \) for all \( i \in [-t_1 + 1, t_2 - 1]_{\mathbb{Z}} - \{0\} \). As an example, at time 0 when \( M \) scans \( \triangleright \) at the start cell and the rest of the cells are \( \square \), we obtain \( t_1 = 0, t_2 = 1, \gamma_0 = \triangleright \), and \( \gamma_{t_2} = \square \).

A session step consists of one of the following series of moves of \( M \): starting at \( \gamma_0 \), (1) write \( \gamma_{\text{new}} \) over \( \gamma_0 \) and make a stationary move, (2) write \( \gamma_{\text{new}} \), move in direction \( d \in \{-1, +1\} \) passing through \( \gamma_i \)'s, reach either \( \gamma_{-t_1} \) or \( \gamma_{t_2} \), and stop, and (3) write \( \gamma_{\text{new}} \), make one step in direction \( d \in \{-1, 0, +1\} \), make a turn, pass through \( \gamma_i \)'s, reach either \( \gamma_{-t_1} \) or \( \gamma_{t_2} \), and stop.

In the following simulation, each symbol \( \gamma_1 \) is translated into a region that contains its corresponding symbol \( \gamma'_1 \).

(*) Meanwhile, we assume that \( \gamma'_0 \) is of the form \( \gamma'_0^{(c,0)} \) for some \( c \in [0, k]_{\mathbb{Z}} \) and that \( \gamma'_0 \) is written at cell, say, \( s \in \mathbb{N} \). The first assumption implies the existence of \( c \) consecutive blank blocks in the left-side area of \( \gamma'_0^{(c,0)} \). It also follows that the current block has the form \( \gamma'_0^{B(01)} \ldots B(0i) \) for some constant \( c \in [0, dv(\gamma_0) - 1]_{\mathbb{Z}} \). In what follows, we also assume that the previous session step of \( M \) is not a storage-stationary move. We argue two cases (I) (II) separately.

(1) Let us consider the first case where the counter head is not activated. Since we do not need to simulate the behaviour of the counter head, it suffices to simulate the next session step of \( M \) using \( N \)'s storage tape.

Consider the case where the storage-tape head of \( M \) came from the left. Clearly, \( \gamma_0 \neq \triangleright \) follows. Assume that we remember the symbol \( \gamma'_1 \) in the form of inner state. In what follows, we examine two cases, (i) and (ii), depending on whether \( \gamma_0 \) is \( \square \) or not.

(i) Assume that \( \gamma_0 \neq \square \). Note that both separator blocks around \( \gamma'_0 \) in the current region are not blank. This makes it possible to discern the borders to the neighbouring regions.

(a) If \( d = 0 \), then \( N \) changes \( \gamma'_0^{(c,0)} \) to \( \gamma'_{\text{new}}^{(c,0)} \) without further moving its storage-tape head.

(b) Assume that \( d = +1 \). There are two more cases to consider. If \( M \) moves rightward to \( \gamma_{t_2} \), then \( N \) overwrites \( \gamma'_0^{(c,0)} \) by \( \gamma'_{\text{new}}^{(c,0)} \) moves rightward by changing \( B(0c) \) to \( B(0c+1) \), and crosses the border to the neighbouring region. From there, \( N \) skips blank regions until entering the first non-blank region, which holds \( \gamma'_1 \). There may be a certain number of blank blocks in this region before reaching \( \gamma'_1 \). Since \( \gamma_{t_2} \neq B, N \) can find this representative \( \gamma'_1 \) in this neighbouring region and stop at the cell containing \( \gamma'_0 \). In contrast, if \( M \) steps to \( \gamma_{1} = B \) and then returns to \( \gamma'_{\text{new}} \), then \( N \) writes \( \gamma_{\text{new}}^{(c,0)} \) over \( \gamma'_0^{(c,0)} \), moves to the right, crosses the border to the neighbouring region, makes a turn at the time of encountering the first \( B \), and returns to \( \gamma'_{\text{new}}^{(c,0)} \). If \( dv(\gamma_0) < k - 1 \), then \( N \) stops here; otherwise, \( N \) continues moving leftward until reaching \( \gamma'_{-t_1} \) and then stops.

(c) When \( d = -1 \), \( N \) writes \( \gamma_{\text{new}}^{(c,0)} \) over \( \gamma'_0^{(c,0)} \), makes a left turn, continues moving leftward until finding then representative \( \gamma'_{-t_1} \), and stops.

(ii) Consider the case where \( \gamma_0 = \square \). Note that all cells located in the right-side area of cell \( s \) hold \( \square \). There are three cases (a) (c) to examine.

(a) If \( d = 0 \), then \( N \) writes \( \gamma_{\text{new}}^{(0,0)} \) over \( \square \) and makes a stationary move. This is possible because \( \gamma_{\text{new}}^{(0,0)} \neq B \).

(b) In contrast, assume that \( d = +1 \). We write \( \gamma_{\text{new}}^{(0,0)} \) over \( \square \) and move to the right. Since we need to secure enough open space for future simulations of the counting head, we wish to generate a new
region. In an early simulation, we have already generated the first half of a region, and thus we need to generate the second half of this region at first. For this purpose, using $1^n$ on the input tape, $N$ moves rightward passing through $(k - e - 1)n$ cells by changing $\square$ to $B^{(1)}$, creates a new border, continues moving for $kn$ cells by changing $\square$ to $B^{(1)}$ to find the centre of the new region, and finally stops. This process newly generates the first half of a region.

(2) Consider the case where the storage-tape head came from the right. The major deviation from (1) is the case of $γ_0 = ?. In this case, if $d = 0$, then $N$ makes a stationary move. If $d = +1$, then $N$ crosses the border to the right neighbouring region, continues skipping blank regions until finding $γ'_t_1$, and then stops.

(II) Next, we want to simulate a single counting process of the counter head of $M$ on the depth-susceptible $k$-sda $N$. Assume that the counter head travels rightward passing through $i$ cells and then returns to the start cell. There are two cases (1)–(2) to consider separately. Note that $γ_0 \neq ?$ since $> \in \Gamma^{(k)}$.

(1) Consider the case where the storage-tape head of $M$ came from the left. In this case, we need to mimic the back-and-forth movement of the counter head as follows. The machine $N$ remembers $γ'_0$ in the form of inner states, modifies it to $B^{(a+1)} (\neq \Gamma^{(k)})$ with $a = dv(γ_0)$, and moves its storage-tape head for $i$ steps to the right as the counter head does, by changing every encountered symbol of the form $B^{(c)}$ with $c \in [0, k]_Z$ to $B^{\text{min}(c+1,k)}$ on its way, provided that $B^{(0)}$ denotes $\square$. After making $i$ steps, it makes a left turn, returns to $B^{(a+1)}$, and writes $B$ over $B^{(a+1)}$. The storage-tape head of $N$ again starts moving rightward for exactly $n$ steps (by reading $1^n$ on the input tape) by changing each $B^{\text{min}(c+1,k)}$ to $B^{\text{min}(c+2,k)}$, and it finally writes $γ'_{new}$ since we create an additional consumed block in the left-side area of $γ'_{new}$.

Finally, if $d = 0$, then the tape head stops here. By contrast, when $d = +1$ (resp., $d = -1$), $N$ moves the storage-tape head rightward (resp., leftward), skipping blank regions until encountering a representative $γ'_{0_r}$ (resp., $γ'_{t_l}$). This case is illustrated as Figure 1.

(2) Assume that $M$ moved to $γ_2$ from the right. Note that $γ_0 \neq ?$. Symmetrically to (1), we generate a new consumed block in the left-side area of $γ'_{0_r}$.

(**) To complete the simulation, let us consider the second case where $γ'_0$ has the form $γ^{(0,e)}_0$ for a certain number $e \in [0, k]_Z$. The current block has the form $B^{(c)} \cdots B^{(c)} γ^0_0$ and there are $e$ consumed blocks in the right-side area of $γ_2^{(0,e)}$. Nonetheless, this case can be symmetrically treated by skipping all consumed blocks as described above.

Finally, we combine all the lemmas (Lemmas 3.2–3.6) and verify Theorem 3.1.

**Proof:** Let $k \geq 2$. The implication (1)$\Rightarrow$(2) is shown as follows. Take any language $L$ over alphabet $Σ$ in LOG$k$SDA. There exist a function $f : Σ^* → Σ_2^*$ in FL for an appropriate alphabet $Σ_2$ and a depth-susceptible $k$-sda $M$ working over $Σ_2$ such that, for any string $x \in Σ^*$, if $x \in L$, then $M$ accepts the string $f(x)$; otherwise, $M$ rejects it. By Lemma 3.2, we can obtain a log-space depth-susceptible aux-$k$-sda $N$ that recognizes $\{x \in Σ^* | f(x) \in L(M)\}$ in polynomial time.

Lemma 3.3 obviously leads to the implication (2)$\Rightarrow$(3). Finally, we want to show that (3) implies (1). Given a language $L$, we assume that there is a polynomial-time $k$-sda$2(\ell)$ $M$ recognizing $L$ for a certain number $\ell \geq 2$. We transform this $k$-sda$2(\ell)$ to another $k$-sda$2(\ell + 1)$ $M'$ by providing a (dummy) counter head. We repeatedly apply Lemma 3.4 to reduce the number of input-tape heads down to 2 by modifying the target language to $L_{ab}$. Lemma 3.5 then implies the existence of a $k$-sda$2(2)$ $N$ with one-way input-tape and counter heads such that $N$ correctly recognizes $L_{rev}$ in polynomial time. By Lemma 3.6, we further obtain a polynomial-time depth-susceptible $k$-sda $K$ that can recognize $L'$, which is of the form $\{x | x \in L_{ab}\}$. Given an input $z$ to $L$, we define $f(z)$ to be the input
\( \tilde{x} \) obtained by running a series of the processes, of Lemmas 3.4–3.5, which reduce the number of input-tape heads. By the clear description of these reduction processes, this function \( f \) is computed in polynomial-time using only log space. Since \( L' = \{ f(x) \mid x \in L \} \) and \( f \in FL \), we conclude that \( L \) belongs to LOGkSDA.

4. Universal simulators and L-m-hard languages

As a major characteristic feature, we intend to prove the existence of concrete, generic L-m-hard languages for LOGkSDA for each index \( k \geq 2 \). For this purpose, we first construct a universal simulator that has an ability to precisely simulate all depth-susceptible \( k \)-sda’s when appropriate encodings of both \( k \)-sda’s and inputs are given. We further force this universal simulator to be a ‘depth-immune’ \( k \)-sda2(4).

4.1. LOGkSDA-hard languages

Sudborough [23] earlier proposed, for every number \( m \geq 2 \), the special ‘tape-hardest’ language \( L_0(m) \), which is L-m-complete for DCFL. This language literally encodes all transitions of deterministic pushdown automata so that we can simulate these machines step by step using a stack. Since \( L_0(m) \) belongs to DCFL, it is also L-m-complete for LOGDCFL because LOGDCFL is closed under L-m-reductions. Sudborough’s success comes from the fact that the use of one-way and two-way deterministic pushdown automata makes no difference in formulating LOGDCFL. In a similar spirit, we propose the following decision problem, MEMBk, for kSDA. Recall that a decision problem is identified with its associated language.

**Membership kSDA Problem (MEMBk):**
- **Instance:** an encoding \( \langle M, x \rangle \) of a depth-susceptible \( k \)-sda \( M \) over alphabet \( \Sigma \) and an input \( x \).
- **Question:** does \( M \) accept \( x \)?

A key to an introduction of MEMBk is a ‘generic’ scheme of encoding both a depth-susceptible \( k \)-sda \( M \) and an input \( x \) into a single string \( \langle M, x \rangle \) over a fixed alphabet, which is independent of the choice of \( M \) and \( x \). In Section 4.2, we will explain such an encoding scheme in detail.

Let us recall the depth-susceptibility condition imposed on \( k \)-sda2(\( \ell \)) in Section 3.2, which is a restriction on the behaviour of the \( k \)-sda2(\( \ell \)) while reading storage-tape symbols in \( \Gamma^{(k-1)} \cup \Gamma^{(k)} \). We remove this condition and introduce a depth-immune \( k \)-sda2(\( \ell \)) in a way similar to the introduction of a depth-immune \( k \)-sda in Section 2.2. With this new model, we define \( k \)-SDA2(\( \ell \)) to be the family of all languages recognized by depth-immune \( k \)-sda2(\( \ell \))’s running in polynomial time.

We assert the following two statements.

**Theorem 4.1:** Let \( k \geq 2 \).
1. The language MEMBk belongs to \( k \)-SDA2(4).
2. MEMBk is L-m-hard for kSDA and thus for LOGkSDA.

To prove Theorem 4.1(1), it suffices in essence to verify the following lemma. For convenience, a \( k \)-sda2(\( \ell \)) \( U \) is called a universal simulator for depth-susceptible \( k \)-sda’s if, for any depth-susceptible \( k \)-sda \( M \) and any input \( x \) given to \( M \), (i) \( U \) takes an input of the form \( \langle M, x \rangle \) and (ii) if \( M \) halts on \( x \), then \( U \) enters the same type (accepting or rejecting) of halting states as \( M \) does; otherwise, \( U \) does not halt.

**Lemma 4.2:** For each \( k \geq 2 \), there exists a special depth-immune \( k \)-sda2(4) that works as a universal simulator for depth-susceptible \( k \)-sda’s.
In Section 4.3, we wish to prove Lemma 4.2 and subsequently Theorem 4.1. The proof of the lemma provides a detailed construction of the desired depth-immune $k$-sda3(4) universal simulator, which takes an encoding $\langle M, x \rangle$ of a depth-susceptible $k$-sda $M$ and its input $x$ and then properly simulates $M$ on $x$ using only a depth-$k$ storage tape of the simulator. Such a proper encoding scheme enables us to construct the desired universal simulator. Theorem 4.1(1) instantly follows from the existence of a universal simulator.

4.2. A desirable encoding scheme

Hereafter, we describe the desired encoding $\langle M, x \rangle$ of a depth-susceptible $k$-sda $M$ and an input $x$. We need to heed special attention to how to encode a pair of $M$ and $x$ into a single string $\langle M, x \rangle$ so that we can easily retrieve $M$ and $x$ from $\langle M, x \rangle$ using a depth-$k$ storage tape for the purpose of the simulation of $M$ on $x$. The desired encoding of $M$ and $x$ needs to keep all the information on the transitions of $M$ intertwined with all bits of $x$ in sequence. However, since the storage usage of $k$-sda’s are quite different from that of deterministic pushdown automata, our encoding scheme is therefore quite different from Sudborough’s scheme.

An underlying idea of Sudborough’s construction of his tape-hardest languages is the notion of cancelling pairs. A similar idea will be used implicitly in the following construction.

Since a $k$-sda uses arbitrary sets $Q$, $\Sigma$, and $\Gamma$, we need to express them using only fixed alphabets independent of $M$ and $x$. In the rest of this subsection, we assume that $M$ has the form $(Q_M, \Sigma_M, \Gamma_M^{(e)})_{e \in [0,k]Z}$, $\delta, p_1, Q_{M,acc}, Q_{M, rej}$ satisfying the following specific conditions: $Q_M = \{p_1, p_2, \ldots, p_m\}$, $\Sigma_M = \Sigma \cup \{\varepsilon, \sigma_1, \sigma_2, \ldots, \sigma_c\}$, and $\Gamma_M = \bigcup_{e \in [0,k]Z} \Gamma_M^{(e)} = \{\gamma_1, \gamma_2, \ldots, \gamma_m\}$ with $\Gamma_M^{(0)} = \{\square\}$ and $\Gamma_M^{(k)} = \{\varepsilon, B\}$. Without loss of generality, we further assume that $Q_{M, acc} = \{p_2\}$ and $Q_{M, rej} = \{p_3\}$. The transition function $\delta$ thus maps $(Q_M - \{p_2, p_3\}) \times \Sigma_M \times \Gamma_M$ to $Q_M \times \Gamma_M \times D_1 \times D_2$. For simplicity, we assume that $c, m_1$, and $m_2$ satisfy that $c = 2^{[\log c]}$, $m_1 = 2^{[\log m_1]}$, and $m_2 = 2^{[\log m_2]}$. Under this assumption, all the elements in $\Sigma_M, Q_M$, and $\Gamma_M$ are precisely expressed as the corresponding elements in $\{a_0, a_1\}^{[\log c]}$, $\{a_0, a_1\}^{[\log m_1]}$, and $\{a_0, a_1\}^{[\log m_2]}$ (using their lexicographic order), respectively.

In what follows, let $e, e' \in [0,k]Z$, $h_1, h_2 \in [m_1]$, $h_3, h_4 \in [m_2]$, $h_5 \in [c]$, $\gamma_1 \in \Gamma_M^{(e)}$, $\gamma_3 \in \Gamma_M^{(e')}$, and $d_1 \in \{0, +1\}$. Notice that the input tape is read by $M$ only once from left to right. Generally, a transition of $M$ has one of the following 7 forms.

(1) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \gamma_3, d_1, +1)$ (moving to the right) if $e < k$ is even and $e' = \min \{e + 1, k\}$.
(2) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \gamma_3, d_1, -1)$ (left turn) if $e < k$ is even and $e' = \min\{e + 2, k\}$.
(3) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \gamma_3, d_1, -1)$ (moving to the left) if $e < k$ is odd and $e' = \min \{e + 1, k\}$.
(4) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \gamma_3, d_1, +1)$ (right turn) if $e < k$ is odd and $e' = \min\{e + 2, k\}$.
(5) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \varepsilon, d_1, +1)$.
(6) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, B, 0, d_2)$ if $d_2 \neq 0$ and either $e = k$ or $e = k - 1$.
(7) $\delta(p_{h_2}, \sigma_{h_3}, \gamma_1) = (p_{h_3}, \gamma_1, d_1, 0)$ (storage-stationary move).

We further set $m = (m_1, m_2)$ and $\theta = m_1 m_2 = \langle Q_M, |\Gamma_M| \rangle$. Moreover, we give two bijections mapping $[m_1] \times [m_2]$ to $[\theta]$ and mapping $[m_1] \times [m_2] \times [m_1] \times [m_2]$ to $[\theta^2]$. For the desired universal simulator, we define a distinguished symbol ‘#’ and we define $\Sigma$ and $\Gamma^{(e)}$, for every $e \in [0,k]Z$, to satisfy $\Sigma = \{a_0, a_1, 0, 1, \#\}$, $\Gamma^{(0)} = \{\square\}$, $\Gamma^{(1)} = \{a_0, a_1, 0, 1, \#\}$, and $\Gamma^{(k)} = \{\varepsilon, B\}$. Finally, we define $\tilde{\Gamma}$ to be $\bigcup_{e \in [0,k]Z} \tilde{\Gamma}^{(e)}$.

We encode an input $x = x_1 x_2 \ldots x_n$ of length $n$ symbol by symbol as follows. Notice that $x_0 = \varepsilon$ and $x_{n+1} = \sqcap$. Let $X_l = x_l \#$ for any position $l \in [0, n + 1]Z$, where $x_l$ is viewed here as a string over $\Sigma$. These strings $X_l$ will be later combined with an encoding of transitions of $M$. 

We then encode the transitions of the forms (1)–(7) in the following fashion. Since all information stored in a storage tape must be modified as a tape head traverses through the tape, we need to prepare multiple copies of all the transition rules.

Let \( h = (h_1, h_2, h_3, h_4) \), \( h' = (h_2, h_4) \), and \( h'' = (h_1, h_2, h_4) \), where \( h_1, h_3 \in [m_2] \) and \( h_2, h_4 \in [m_1] \). Moreover, let \( h_5 \in [c] \) and \( d_1 \in \{0, +1\} \). By treating \( h \) as an element of \([m_1] \times [m_2] \times [m_1] \times [m_2] \), we identify it with a number in \([\theta^2] \). Similarly, \( h' \) is identified with a number in \([m^2_1] \). Abusing the notations, we further assume that \( h \in [0^2] \) and \( h' \in [m^2_1] \). We encode the transitions (1)–(7) into six types of strings \( \tilde{u}_i^{(+)}, \tilde{u}_i^{(-)}, \tilde{u}_i^{(l)}, \tilde{u}_i^{(r)}, \tilde{u}_i^{(0)} \), and \( \tilde{u}_i^{(d)} \), which are defined below, where the superscripts ‘lt’ and ‘rt’ respectively refer to ‘left turn’ and ‘right turn.’ Those strings are later referred to as encoded transitions. Hereafter, for convenience, \( p_n, \sigma_l, \) and \( \gamma_l \) are assumed to be already translated. The items (1’)–(7’) below directly correspond to the transitions (1)–(7). For readability, nonetheless, we omit all the supplemental conditions stated in (1)–(7). We set \( \tilde{a} = 100,000,001 \) if \( \tilde{d} = -1, 0, +1 \), respectively.

\[
\begin{align*}
(1') & \quad \tilde{u}_{i,0}^{(+)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 001 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. \\
(2') & \quad \tilde{u}_{i,0}^{(l)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 100 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. \\
(3') & \quad \tilde{u}_{i,1}^{(-)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 001 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. \\
(4') & \quad \tilde{u}_{i,1}^{(r)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 001 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. \\
(5') & \quad \tilde{u}_{i,2}^{(+)} \equiv \sigma_{h_5} \# p_{h_2} \# p_{h_1} \# \tilde{d}_2 \# B^c \# p_{h_4} \# 10. \\
(6') & \quad \tilde{u}_{i,2}^{(d)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 000 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. \\
(7') & \quad \tilde{u}_{i,3}^{(0)} \equiv \sigma_{h_5} \# h_1 \# p_{h_2} \# 001 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1}. 
\end{align*}
\]

Let us define \( P^{(c)} \) as follows. For any \( j \in \{h, h', h''\} \) and any \( e \in \{0, 1, 2, 3\} \), the segment \( \sigma_{h_5} \# h_{l} \# p_{h_2} \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1} \) of \( \tilde{u}_{j,e} \) is called a receptor of \( \tilde{u}_{j,e} \) and the segment \( 001 \# (\gamma_{h_3} \#)^0 \# p_{h_4} \# 1^{d_1} 10^{1-d_1} \) of \( \tilde{u}_{j,e} \) is called a residue of \( \tilde{u}_{j,e} \). We set \( \tilde{u}_{0}^{(0)} \) to be \( \tilde{u}_{1,0}^{(0)} \# \tilde{u}_{0,0}^{(0)} \# \ldots \# \tilde{u}_{0,0}^{(0)} \# \tilde{u}_{0,1}^{(0)} \# \tilde{u}_{0,2}^{(0)} \# \ldots \# \tilde{u}_{0,2}^{(0)} \# \tilde{u}_{m_2}^{(0)} \). Collectively, we set \( P^{(+)} \) to be \( \tilde{u}_{0}^{(0)} \# \tilde{u}_{2}^{(0)} \). The other three strings \( P^{(l)}, P^{(-)}, \) and \( P^{(r)} \) are defined similarly. Combining those four strings, we define \( P \) to be \( P^{(+)} \# P^{(l)} \# P^{(-)} \# P^{(r)} \).

Let \( S \) denote the string \( p_1 \# p_2 \# \ldots \# p_m \# \) and let \( G \) be \( \gamma_1 \# \gamma_2 \# \ldots \# \gamma_m \# \). Furthermore, we set \( X \) to be \( X_0 X_1 \ldots X_{n+1} \). Finally, the encoding \( \langle M, x \rangle \) of \( M \) and \( x \) is defined as the string \( X \# S \# G \# P \# \). Note that the length of \( \langle M, x \rangle \) is bounded by \( O((|Q_M||\Gamma_M||\Sigma_M||x|)^4) \). It is not difficult to see by the definition that the encoding \( \langle M, x \rangle \) is uniquely determined from \( M \) and \( x \).

### 4.3. Proofs of Theorem thm4.1 and Lemma thm4.2

Our goal of this subsection is to provide the proofs of Theorem 4.1 and Lemma 4.2. For this purpose, we first present, given an arbitrary encoded string \( \langle M, x \rangle \), how to simulate \( M \) on \( x \) precisely using a depth-\( k \) storage tape of a universal simulator.

We first introduce necessary terminology. A storage-tape content refers to a sequence of symbols written on the storage tape from the start cell to the leftmost tape cell containing \( \square \), say, cell \( t \). For our convenience, we call a string \( \xi \) of the form \( 0^k \sigma_0 \# 0 \ldots \sigma_{l-1} \# 0 \sigma_l \# 0 \sigma_{l+1} \ldots \sigma_t \) a storage configuration of \( M \) if \( M \) is scanning cell \( b \) of the input tape, the storage-tape head of \( M \) is scanning cell \( l \), \( \sigma_0 \) equals \( \triangleright \), \( \hat{\sigma}_l \) is of the form \( (1^k, \sigma_l) \), and the string \( \sigma_0 \ldots \sigma_{l-1} \sigma_{l+1} \ldots \sigma_t \) is a storage-tape content of \( M \).

Here is a key idea behind the following proof of Lemma 4.2. A storage tape is used to keep track of the information on the current surface configuration. To simulate the next move, we go through all encoded transitions on an input tape by way of removing any ‘uncancelled pair’ from the storage-tape to locate the target transition to be taken at the next move. If we successfully delete the old surface
configuration, we write a new surface configuration into an 'appropriately chosen' new section of the storage tape. The last process is needed because our storage tape is of depth \( k \), and thus we cannot place a new symbol at an arbitrary location of the storage tape.

**Proof:** Let \( k \geq 2 \). We intend to construct a depth-immune \( k\)-sda\(_2(4) \) universal simulator working for all depth-susceptible \( k\)-sda’s. Let \( M = (Q_M, \Sigma_M, \{ \Gamma^e_M \}_{e \in \{0,1\}^+}, \theta, P_M, M_{acc}, M_{rej}) \) denote any depth-susceptible \( k\)-sda and let \( x = x_1x_2 \ldots x_n \) be any input of length \( n \) given to \( M \). Additionally, we assume \( x_0 = \rightarrow \) and \( x_{n+1} = \leftarrow \). Let us recall the notations, \( \Sigma, \Gamma^e, \Gamma^c, m_1, m_2, \) and \( \theta \), defined in Section 4.2, associated with \( M \). Let \( \alpha = ([\log m_2] + 1)\theta \). Assume that \( \langle M, x \rangle \) has the form \( XS\#G\#P\# \) with \( X, S, G, \) and \( P \) given in Section 4.2.

Into our storage alphabet, we include \( k - 1 \) different symbols \( B_1, B_2, \ldots, B_{k-1} \) satisfying \( dv(B_i) = i \) for any \( i \in [k-1] \) to express 'intermediate' blank and we reserve \( B \) for the frozen blank symbol. For the ease of the description of later simulation, whenever we read \( B_i \) for each \( i \in [k-1] \), we automatically replace it with \( B_{i+1} \), where \( B_k \) is understood as \( B \).

To represent the content of the depth-\( k \) storage tape of \( M \), we use a 'block' of two 'sections'. Each section is a string of the form \( (\sigma \#)^d \) or \( B_i^a \) with \( \sigma \in \Gamma \) and \( i \in [k] \), and an entire block has the form \( (\sigma \#)^d B_{2+i}^a, B_{3+i}^a (\sigma \#)^d, \) or \( B_{2+i}^a \) with \( i < k - 1 \) and \( j < \frac{k-2}{2} \). Notice that each block has length exactly \( 2a \) since \( |\sigma \#| = [\log m_2] + 1 \). The string \( (\sigma \#)^d \) is called the core of this block if any, and \( \sigma \) is called the core value. The depth of a block is the depth of its core value. For simplicity, we say that a block is currently accessed if the storage-tape head is stationed in this block. Similarly, \( \alpha, \psi, p_h, \) and \( \gamma \) are said to be currently accessed if input-tape heads are reading them. For technical reason, we further allow a block to be \( \square \) and \( \Rightarrow \). The tape content can be expressed as a series \( Y_0B_0Y_1B_1 \ldots Y_tB_t \) of blocks separated by \( B \) with \( Y_0 = \Rightarrow \), where cell \( i \) is the leftmost initially-blank cell. We denote this sequence by \( Y \). This is in part illustrated in Figure 2. We next decode this sequence \( Y \) into a storage configuration \( y_0y_1 \ldots y_t \) as follows. If \( Y_i \) is \( \Rightarrow \) or \( \square \), then we simply set \( y_i = \Rightarrow \) and \( y_i = \square \), respectively. If \( Y_i \) is frozen blank (i.e. \( Y_i = B_{2a} \)), then we set \( y_i = B \). Assume that \( Y_t \neq \{ \square, \Rightarrow \} \) and the core value of \( Y_t \) is \( \sigma \). If we currently access \( Y_t \) and \( p_h \), then we set \( y_i = (1^h, \sigma) \), which indicates that \( M \) is in inner state \( p_h \) and \( \sigma \) appears in the cell of \( M \)'s storage-tape head location \( l \). In contrast, if \( Y_t \) equals \( (\sigma \#)^d \) but \( Y_t \) is not currently accessed, then we set \( y_t = \sigma \).

In what follows, we want to demonstrate by induction on the number \( r \) of steps that \( \langle M, x \rangle \) correctly 'generates' the \( r \)th storage configuration of \( M \) on \( x \). To express \( Y \) at step \( r \), we use the notation of \( Y(r) \) for clarity. Similarly, we write \( Y_t(r) \) for \( y_t \) at step \( r \). Initially, at time \( t = 1 \), we produce \( Y(0) = Y_0(B_0B_0) \) on the storage tape, where \( Y_0(0) = \Rightarrow \) and \( Y_1(0) = \square \). Our goal is to verify that the following statement \((*) \) is true.

\((*) \) For each index \( r \geq 1 \), if \( M \) scans \( x_r \) and \( Y \) in inner state \( p \) and makes a transition at time \( r \), then we read \( X_r \) in \( X, p \) in \( S \), and \( Y \) in \( G \), find the corresponding encoded transition, simulate the \( r \)th step of \( M \) by executing this encoded transition, and produce \( Y(r) = Y_0(r)B_0(r)B \ldots Y_t(r) \) on the depth-\( k \)

---

**Figure 2.** Blocks on the storage tape and the tape head positions at steps \( r - 1, r, \) and \( r + 1 \), where \( t, h, s_1, s_2 \in [m_2] \) and \( e + 2 \leq k \). The tape head moves to the right in (1)–(2) and to the left in (3)–(4) by making a left turn in (3).
storage tape, where cell $t$ is the leftmost initially-blank cell. The sequence $y = y_0^{(r)} y_1^{(r)} \ldots y_t^{(r)}$ decoded from $Y^{(r)}$ then matches the storage configuration of $M$ on $x$ at time $r$.

Meanwhile, we assume that ($\ast$) is true. After processing $X_{n+1}$, since the storage configuration of $M$ must contain a halting state, we check whether this halting state is indeed $p_2$, which indicates that $M$ is in an accepting state. If so, we accept the input $(M,x)$; otherwise, we reject it. Hereafter, we focus on proving ($\ast$) by induction on $r \in \mathbb{N}^+$. We wish to construct the desired $k$-sda2(4) universal simulator, say, $U$. For convenience, we call four input-tape heads of $U$ by heads 1–4, where head 1 reads encoded transitions in the list $P$, head 2 ‘points’ the encoding of the current inner state $p$ in the list $S$, head 3 accesses $X_r$ to retrieve $x_s$, and head 4 points the encoding of the current storage symbol $\gamma$ in $G$. During the following simulation, we remember the depth of the neighbouring block from which the storage-tape head comes to the currently accessed block.

For readability, we purposely allow $U$ to change any symbol not in $\Gamma^{(k)}$ into $B$ in a single step.

(Step $r = 1$) Recall that $X_0 = \triangleright \#$. At step 1, since $M$ applies a transition of the form $\delta(p_1, \triangleright, \triangleright) = (p_{h_4}, \triangleright, d_1, +1)$ given in the item (5) in Section 4.2, its storage-tape head moves from cell 0 to cell 1, and thus the storage configuration of $M$ on $x$ at step 1 is $\triangleright \left(1^{h_4}, \Box \right)$. In scanning $\triangleright$ on the storage tape and $p_1$ in the list $S$, we search $P$ for $\tilde{u}_{h_4}^{(+)} = (\triangleright \# \triangleright p_1 \#001 \# \triangleright p_{h_4} \#1 \triangleright d_1 + 1 \triangleright 0^{1-d_1} \#)$ with $h_4 = (1, h_4)$. We move the storage-tape head to the right to access the first encountered $\Box$. From the current tape content, it follows that $Y_1^{(0)} = \triangleright, Y_1^{(1)} = \Box^{2a}$, and $Y_1^{(k)} = Y_1^{(0)}$ for any other indices $l$. Thus, we obtain $Y_1^{(0)} = \triangleright$ and $Y_1^{(l)} = (1^{h_4}, \Box)$. Clearly, the resulting sequence $y_1^{(l)}$ matches the storage configuration of $M$ on $x$ at step 1.

(Step $r \geq 2$) Let us consider the $r$th step of $M$. Assume by induction hypothesis that ($\ast$) is true for $Y^{(r-1)}$ at step $r - 1$. Take an arbitrary number $l \in [0, n+1]_{\mathbb{Z}}$ and assume that $Y_l^{(r-1)}$ is of the form $(\gamma_1, \#)^0 B_{i+1}^g (\gamma_1, \#)^0$, or $B_i^{a\#}$ with $i < \frac{k-1}{2}$ and $j < \frac{k-2}{2}$. If $Y_l^{(r-1)}$ is not currently accessed by $U$, then $Y_l^{(r)}$ equals $Y_l^{(r-1)}$. In the following argument, we therefore assume otherwise. Assume also that head 2 resides over the string $p_{h_2}$ in $S$, indicating that $M$ is in inner state $p_{h_2}$. Assume further that head 3 is reading $X_r$ that contains $x_s$, where $s \in [0, n + 1]_{\mathbb{Z}}$. This means that $M$’s tape head rests on cell $s$, which holds the input symbol $x_s$. At step $r$, $M$ makes one of the transitions (1)–(7) given in Section 4.2. In what follows, we separately deal with those transitions.

(1) Assume that $M$ applies a transition of the form $\delta(p_{h_2}, \sigma_{h_2}, \gamma_1) = (p_{h_4}, \gamma_3, d_1, +1)$ with $\sigma_{h_2} = x_s$, $\gamma_1 \in \Gamma^{(l)}$, and $\gamma_{h_3} \in \Gamma^{(\min(e+1, k))}$ for an even number $e < k$. In the currently accessed block, say, $Y_{i+1}^{(r-1)}$, the storage-tape head is scanning $B, \Box$, or $\triangleright$. We then discuss two cases (i) and (ii) separately.

(i) In the first case of $\min(e + 1, k) < k$, we need to cope with the following subcases (a) and (b).

(a) Assuming that $\gamma_{h_1} \not\in \{\triangleright, \Box\}$, we first find the location of the string $\gamma_{h_1}$ in $G$. Since the currently accessed block $Y_{i+1}^{(r-1)}$ is of the form $(\gamma_1, \#)^0 B_c^g$, we move the storage-tape head rightward through the first section $(\gamma_1, \#)^0$. We move head 4 through $G$ by picking substrings $\gamma^\#$ one by one. As we read one string of the form $\gamma_1^\#$ in the section, we check whether $\gamma$ matches $\gamma_{h_1}$ by simultaneously comparing between $\gamma^\#$ and $\gamma_{h_1}^\#$ symbol by symbol. After each comparison, we mark $\gamma_{h_1}^\#$ as ‘being read’ by changing it into $B_{c+1}^{[\log c]+1}$. Even in the case of discovering any mismatch in the middle of $\gamma_{h_1}^\#$, we continue modifying the content of tape cells until we completely change this entire string $\gamma_{h_1}^\#$ to $B_{c+1}^{[\log c]+1}$, and we then pick another substring and repeat the above procedure. This search is possible because we have an enough number of copies of the string $\gamma_{h_1}^\#$ in $Y_{i+1}^{(r-1)}$, and thus we eventually find $\gamma_{h_1}^\#$ in $G$.

Since heads 2–4 correctly point the locations of $x_s$, $p_{h_2}$, and $\gamma_{h_1}$, we can look for the correctly encoded transition $\tilde{u}_{h_0}^{(+)}$ of the form $x_0^\# \gamma_{h_1}^\# p_{h_2}^\# 001^\# (\gamma_{h_3}^\#)^0 \# p_{h_4}^\# 1^{d_1+1} 0^{1-d_1} \#$ in $P$ by conducting the following search for the receptor $x_0^\# \gamma_{h_1}^\# p_{h_2}^\#$ of $\tilde{u}_{h_0}^{(+)}$. In search of $\tilde{u}_{h_0}^{(+)}$, we first move head 1 to the first symbol of $P$ and, by moving head 1 through $P$, we pick encoded transitions, say, $\tilde{u}$ one by
one. We then examine that \((x_s, p_{h_2}, \gamma_{h_1})\) truly appears in the receptor of \(\tilde{u}\). If not, we continue picking another transition. This recursive process does not require any movement of \(U\)'s storage-tape head.

Once we find \(\tilde{u}_h^{(+)}\) by scanning its residue 001#(\(\gamma_{h_3}\)#\(\gamma\)\(p_{h_4}\)#\(1^{d_1+1}0^{1-d_1}\)#, we overwrite the second section \(B_2^e\) of \(Y^{(r-1)}\) by \((\gamma_{h_3}\)#\(\gamma\) symbol by symbol until reaching the frozen blank symbol \(B\), which acts as a block separator. We then obtain the newly modified block \(Y^{(r)}\), which has the form \(B_2^{e+1}(\gamma_{h_3}\)#\(\gamma\). Note that all the other blocks are intact. We further move the storage-tape head rightward to the first non-\(B\) symbol. See Figure 2(1)--(2) for an illustration of storage-tape head moves. Finally, we update the locations of heads 2 and 3 by moving head 2 to \(p_{h_4}\), head 3 to \(X_{s+d_1}\), and head 4 to the first symbol of \(P\).

From all the blocks at time \(r\), we obtain \(y_{i}^{(r)}=y_{h_2}, y_{i+1}^{(r)}=(1^{h_4}, \xi)\) if \(y_{i-1}^{(r)}=\xi\), and \(y_{j}^{(r)}=y_{j}^{(r-1)}\) for all other indices \(j\). By the definition, the resulting string \(y^{(r)}=y_{1}^{(r)}y_{2}^{(r)}\ldots y_{i}^{(r)}\) matches the storage configuration of \(M\) on \(x\) at step \(r\).

(b) Consider the next case of \(\gamma_{h_1} = \square\). Since \(\gamma_{h_1} = \square\), we can search for \(\gamma_{h_1}\) in \(G\) by making only storage-stationary moves. We then conduct the aforementioned receptor search to locate \(\tilde{u}_h^{(+)}\) in \(P\). Since the currently accessed block \(Y^{(r-1)}\) is \(\square\), we write \(B_2^e(\gamma_{h_3}\)#\(\gamma\) into this block and move the tape head to the first encountered \(\square\). We also move head 2 to \(p_{h_4}\) and head 3 to \(X_{s+d_1}\).

(ii) Consider the second case of \(\min\{e+1, k\} = k\). Notice that \(\gamma_{h_1} \neq \square\). In this case, we write \(B\) instead of \((\gamma_{h_3}\)#\(\gamma\) in (i), and \(Y^{(r)}\) thus becomes \(B_2^e\).

(2) Consider the case where \(M\)'s transition is of the form \(\delta(p_{h_2}, \sigma_{h_2}, \gamma_{h_1})=(p_{h_4}, \gamma_{h_3}, d_1, -1)\) with \(\sigma_{h_5}=x_s, \gamma_{h_1}\in \Gamma(e)\), and \(\gamma_{h_3}\in \Gamma^{\min\{e+2\}}\) for an even number \(e < k\). Notice that \(x_s \neq \sigma\).

(i) Assume that \(\min\{e+2, k\} < k\) with \(\sigma_{h_5} = x_s\). Notice that the currently accessed block \(Y^{(r-1)}\) is of the form \((\gamma_{h_3}\)#\(\gamma\)\(B_2^e\). Firstly, we try to find \(\gamma_{h_1}\) in \(G\) by simultaneously moving head 4 and the storage-tape head from left to right over the first section \((\gamma_{h_3}\)#\(\gamma\) of \(Y^{(r-1)}\). After this search, the section \((\gamma_{h_3}\)#\(\gamma)\) becomes \(B_2^{e+1}\). We then search \(P\) for \(\tilde{u}_h^{(r)}(\equiv \sigma_{h_5}\#\gamma_{h_1}\#p_{h_2}\#100\#(\gamma_{h_3}\)#\(\gamma\)\#\(p_{h_4}\)#\(1^{d_1+1}0^{1-d_1}\)#) by the aforementioned receptor search. Once we find \(\tilde{u}_h^{(r)}\), we locate the residue 100\#(\(\gamma_{h_3}\)#\(\gamma\)\#\(p_{h_4}\)#\(1^{d_1+1}0^{1-d_1}\)#.

Next, we overwrite the second section of \(Y^{(r-1)}\) by \(\tilde{B}^e\), where \(\tilde{B}\) is a fresh storage symbol. When we reach a block separator, we make a left turn and then change \(B_2^e\) to \(B_2^{e+2}\) from right to left. We continue overwriting the next section \(B_2^{e+1}\) by \((\gamma_{h_3}\)#\(\gamma)\) until reaching a block separator. In the end, we step to the left. Since \(Y^{(r)}_{i-1}=Y^{(r-1)}_{i-1}\) and \(Y^{(r)}_{i}=Y^{(r-1)}_{i}B_2^{e+2}\), we obtain \(y^{(r)}_{i}=\gamma_{h_3}\) and \(y^{(r)}_{i-1}=(1^{h_4}, \xi)\) if \(y^{(r-1)}_{i-1}=\xi\). All the other \(y^{(r)}_{j}\) are updated to be \(y^{(r)}_{j}\). The resulting sequence \(y^{(r)}\) clearly matches the storage configuration of \(M\) on \(x\) at step \(r\). Figure 2(3)--(4) illustrate this process.

(ii) In the case of \(\min\{e+2, k\} = k\), since \(\gamma_{h_3}\) is \(B\), we write \(B^e\) instead of \((\gamma_{h_3}\)#\(\gamma)\) in (i). As a result, \(Y^{(r)}\) becomes \(B_2^{2e}\).

(3) This case is symmetric to (1). Since \(M\)'s storage-tape head earlier came from the right, \(U\)'s storage-tape head starts at the rightmost symbol of the second section of \(Y^{(r-1)}\). During the search for \(\gamma_{h_1}\) in \(G\), we need to move the storage-tape head leftward. After finding \(\tilde{u}_h^{(r)}\) by the receptor search, we overwrite the first section of \(Y^{(r-1)}\) by \((\gamma_{h_3}\)#\(\gamma)\) from right to left.

(4) This case is handled symmetrically to (2) in a way similar to (3).

(5) In the case of \(\triangleright\), we look for \(\tilde{u}_h^{(+)}\) by conducting the aforementioned search for its receptor \(x_s\#x_2\#p_{h_2}\#\). Since \(d_\triangleright(k) = k\), we do not need to overwrite \(\triangleright\) on the storage tape.

(6) Consider the case where \(M\)'s transition is of the form \(\delta(p_{h_2}, \sigma_{h_2}, \gamma_{h_1})=(p_{h_4}, B, 0, d_2)\) with \(e \in \{k-1, k\}\) and \(d_2 \in \{-1, +1\}\). We discuss two subcases of \(e = k-1\) and \(e = k\) separately.

(i) Assume that \(e = k-1\). In the case where \(e\) is even, \(M\)'s storage-tape head earlier came from the left. Note that the other case where \(e\) is odd is symmetric. By scanning the first section of \(Y^{(r-1)}_j\), we look for \(\gamma_{h_1}\) in \(G\) as we overwrite this section by \(B^2\) until we encounter the second section \(B_2^{k-1}\). Once
we find \( \gamma_{h_1} \) in \( G \), we conduct the receptor search for \( \tilde{u}^{(d_2)}_{h',2} \) and then retrieve its residue to discover the value of \( d_2 \).

Since we remember the depth of the left adjacent block, let \( g \) denote this depth. Consider the first case of \( g < k \). We change the second section \( B^g_{k-1} \) by \( B^g \) and reach a block separator. When \( d_2 = -1 \), we make a left turn and move leftward to the first non-\( B \) symbol. In the case of \( g = k \), on the contrary, if \( d_2 = +1 \), then we continue overwriting the section \( B^g_{k-1} \) by \( B^g \) until reaching a block separator. We then step to the next block. Finally, when \( d_2 = -1 \), based on the fact that the left adjacent cell is frozen blank, without moving the storage-tape head, we can determine whether, at the next step \( r + 1 \), \( M \) makes a right turn or moves further to the left.

(a) Assume that \( M \) make a right turn at step \( r + 1 \). We first overwrite \( B^g_{k-1} \) by \( B^g \) until reaching a block separator. We update the location of head 2 from \( p_{h_2} \) to \( p_{h_3} \), update the depth of the neighbouring block to \( k \), and move to the first symbol of the next block.

(b) If \( M \) moves further to the left at step \( r + 1 \), then we overwrite \( B^g_{k-1} \) by \( B^g \) until reaching a block separator. We then make a left turn, move head 2 to \( p_{h_4} \) from \( p_{h_3} \), and move the storage-tape head leftward to the first non-\( B \) symbol. Since \( M \) is depth-susceptible, \( M \) does not change its behaviour while reading \( B \)s, and thus we can smoothly pass through all blank blocks without any further information.

(ii) On the contrary, if \( e = k \), then \( h_2 = h_4 \) follows. If \( g = k \), then \( d_2 \) must be +1. Since \( M \) does not change its inner state and head direction, we move rightward until encountering the first non-\( B \) symbol. In the case of \( g < k \), it is possible to look for \( \tilde{u}^{(d_2)}_{h_2,4} \) by conducting the receptor search and discover \( d_2 \). If \( d_2 = +1 \), then we move rightward to the first non-\( B \) symbol. In contrast, when \( d_2 = -1 \), we skip the scanning of \( Y^{(r-1)}_i \) and make a left turn to the first non-\( B \) symbol.

(7) Assume that \( M \) makes a storage-stationary move with \( \gamma_{h_1} \in \Gamma^{(c)} \). We first look for \( \gamma_{h_1} \) in \( G \) by reading \((\gamma_{h_1},#)\) in \( Y^{(r-1)}_i \) either from left to right or from right to left, depending on the value of \( e \). After reading it, it is overwritten by \( B^{d+1}_{e+1} \). We then search for the receptor of the form \( x,\#\gamma_{h_1},\#h_{h_2},\# \) to locate \( \tilde{u}^{(0)}_{h_2,4} \) in \( P \). After finding \( \tilde{u}^{(0)}_{h_2,4} \), we retrieve the residue of the form \( 000#(\gamma_{h_1},#)^0p_{h_4}\#1d_1+101−d_1,# \).

Since \( M \)'s storage-tape head does not move, we stop \( U \)'s storage-tape head in the middle of the current block just after finishing the replacement of \((\gamma_{h_1},#)\) by \( B^{d+1}_{e+1} \). Note that, if \( M \)'s input-tape head moves, then \( \gamma_{h_1} \notin \Gamma^{(k-1)} \cup \Gamma^{(k)} \) follows because, otherwise, we obtain \( d_1 = 0 \), a contradiction. When \( M \)'s input-tape head does not move, nonetheless, \( \gamma_{h_1} \neq B \) follows because, otherwise, \( M \) cannot halt. Note that, as long as \( M \) makes storage-stationary moves, we reuse the obtained information on \( \gamma_{h_1} \) without further moving \( U \)'s storage-tape head. When \( M \) eventually moves its storage-tape head either to the right or to the left, \( M \) must take one of the transitions (1)–(6). We then simulate the transition of \( M \) in a way explained in (1)–(6) except for the first process of finding \( \gamma_{h_1} \) in \( G \) since this process has already been done.

Finally, we note that the constructed universal simulator \( U \) is indeed a depth-immune \( k \)-sda(3). This completes the proof.

To close Section 4, we still need to prove Theorem 4.1.

**Proof:** (1) This directly comes from the existence of a depth-immune \( k \)-sda(4) universal simulator for all depth-susceptible \( k \)-sda's, guaranteed by Lemma 4.2.

(2) Let \( M = (Q, \Sigma, \{\Gamma^{(c)}_{e} \}_{e \in [0,k] \cup \{\overset{\rightarrow}{},\overset{\leftarrow}{}, \delta, q_{0}, Q_{\text{acc}}, Q_{\text{rej}} \}) \) be any depth-susceptible \( k \)-sda. We want to show that \( L(M) \) is \( L \)-m-reducible to \( \text{MEMB}_k \). Take any input \( x \) and consider the encoding \( \langle M, x \rangle \) of \( M \) and \( x \), where \( \langle M, x \rangle \) has the form \( X##G#P# \) as defined in Section 4.2. Since \( Q, \Sigma, \) and \( \Gamma \) are all fixed finite sets independent of \( x \), the construction of \( \langle M, x \rangle \) from \( x \) requires polynomial time using only logarithmic space. We define \( f_M(x) = \langle M, x \rangle \) for any \( x \in \Sigma^* \). By the definition of \( \text{MEMB}_k \), it follows that \( M \) accepts \( x \) if \( f_M(x) \) belongs to \( \text{MEMB}_k \). Therefore, we conclude that \( L(M) \) is \( L \)-m-reducible to \( \text{MEMB}_k \) via \( f_M \). Since \( \text{LOGkSDA} \) is closed under \( L \)-m-reductions, \( \text{MEMB}_k \) is also \( L \)-hard for \( \text{LOGkSDA} \). ■
5. A complexity upper bound of kSDA_{imm}

We have discussed the mathematical model of depth-susceptible k-sda’s in Sections 3–4. We here turn our interest to another model, depth-immune k-sda’s, and discuss the computational complexity of kSDA_{imm}. Concerning the complexity of DCFL, Cook [4] earlier demonstrated that DCFL is included in SC^2. In what follows, for any \(k \geq 2\), we intend to present a non-trivial SC^k-upper bound on the complexity of kSDA_{imm}, namely, kSDA_{imm} \subseteq SC^k. Since kSDA_{imm} properly includes DCFL, this upper bound of kSDA_{imm} significantly extends Cook’s result in [4].

**Theorem 5.1:** For any integer \(k \geq 2\), kSDA_{imm} \subseteq SC^k. Thus, \(\omega_{SDA_{imm}} \subseteq SC\) follows.

Since SC^k is closed under L-m-reductions, Theorem 5.1 instantly yields the following corollary.

**Corollary 5.2:** For any \(k \geq 2\), LOGkSDA_{imm} \subseteq SC^k.

Another immediate consequence of Theorem 5.1 is a complexity upper bound of Hibbard’s deterministic k-limited automata (k-lda’s) because k-lda’s (with the blank-skipping property [25]) can be easily simulated by depth-immune k-sda’s by pretending that all input symbols are written on a storage tape. Notice that, in the past literature, no upper bound except for CFL has been shown for Hibbard’s language families [12]. Thus, this is the first time to show non-trivial upper bounds for those families.

**Corollary 5.3:** For any \(k \geq 2\), all languages recognized by Hibbard’s k-lda’s are in SC^k.

To verify Theorem 5.1, we attempt to employ a divide-and-conquer argument to simulate the behaviours of each depth-immune k-sda in polynomial time using only \(O(\log^k n)\) space. Our simulation procedure is based on [1] but expanded significantly to cope with more complex moves of depth-immune k-sda’s.

### 5.1. Markers and contingency trees

We first need to lay out a basic framework to describe the desired procedures of simulating all depth-immune k-sda’s.

Let \(M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{acc}, Q_{rej})\) denote any depth-immune k-sda. We fix an arbitrary input \(x \in \Sigma^*\) and set \(n = |x|\). Since \(M\) halts in polynomial time, we choose an appropriate polynomial \(p\) so that, for any string \(x\), \(p(|x|)\) upper-bounds the running time of \(M\) on the input \(x\). To simulate the behaviour of \(M\) on \(x\), we introduce an important notion of ‘marker’. Any computation of a depth-immune k-sda is characterized by a series of such markers. A marker \(C\) is formally a quintuple \((q, l_1, l_2, \sigma, r, t)\) that indicates the following circumstances: at time \(t\) with section time \(r\) (which will be explained later), \(M\) is in inner state \(q\), its input-tape head is located at \(l_1\), and the storage-tape head is at the \(l_2\)th cell, which contains symbol \(\sigma\). To express the entries of \(C\), we use the following specific notations: \(\text{state}(C) = q\), \(\text{in-loc}(C) = l_1\), \(\text{st-loc}(C) = l_2\), \(\text{symb}(C) = \sigma\), \(\text{sectime}(C) = r\), and \(\text{time}(C) = t\). Let \(M_{\sigma}\) denote the set of all markers of \(M\) on \(x\). To emphasize ‘\(l_2\)’, in particular, we often call \(C\) an \(l_2\)-marker if \(\text{st-loc}(C) = l_2\). Similarly, we call \(C\) a \(t\)-marker if \(\text{time}(C) = t\).

Assume that \(C_t\) is a marker of the form \((q, l_1, l_2, \sigma, r, t)\) at time \(t\) and that \(M\) executes a transition of the from \(\delta(q, x_{l_1}, \sigma) = (p, \tau, d_1, d_2)\) with \(d_1 \in [0, +1]\) and \(d_2 \in [-1, 0, +1]\). We then set \(\text{next-state}(C_t) = p\), \(\text{next-loc}(C_t) = l_2 + d_2\), and \(\text{next-symb}(C_t) = \tau\). To obtain the next marker \(C_{t+1}\) at time \(t + 1\) from \(C_t\), we need to know the current content of cell \(l_2 + d_2\), say, \(\xi\). To obtain this symbol \(\xi\), we first compute the most recent \((l_2 + d_2)\)-marker \(C'\) with \(\text{time}(C') < t\) and then set \(\xi = \text{next-symb}(C')\). The quintuple \((p, l_1 + d_1, l_2 + d_2, \xi, r + |d_2|, t + 1)\) thus becomes the desired marker \(C_{t+1}\).
We then introduce additional critical notions. Let \( C = (q, l_1, l_2, \sigma, r, t) \) and \( C' = (q', l_1', l_2', \sigma', r', t') \) be two arbitrary markers of \( M \) on \( x \). We say that \( C' \) is left-visible from \( C \) if (i) \( l_2' < l_2 \) and (ii) there is no other marker \( C \) satisfying both \( st-loc(C) \leq l_2' \) and \( t' + 1 \leq time(C) < t \). Moreover, \( C' \) is the left-cut of \( C \) if \( C' \) is the most recent left-visible marker from \( C \); that is, \( C' \) is left-visible from \( C \) and \( t' \) is the largest number satisfying \( t' < t \). If the storage-tape head moves to the left from cell \( l_2 \) at time \( t \), then the left cut of \( C_t \) provides the latest content of cell \( l_2 - 1 \). This makes it possible to determine the renewed content of cell \( l_2 - 1 \) by applying \( \delta \) directly. In symmetry, we can define the notions of right-visibility and right-cut.

A 0-section consists of all markers indicating either (a) a right/left turn (i.e. leftmost/rightmost point) or (b) a non storage-stationary move to the right/left followed by a (possibly empty) series of consecutive storage-stationary moves. For any \( d \in \mathbb{N} \), a \((d + 1)-section\) is the union of two consecutive \( d \)-sections. The section time of a marker \( C \) is the total number of 0-sections before \( C \) (not including the 0-section containing \( C \)).

Given a set of markers, a marker \( C \) is said to be the leftmost marker (resp., the rightmost marker) if \( st-loc(C) \) is the smallest (resp., the largest) among the markers in the given set. The leftmost (resp., rightmost) marker in each \( d \)-section \( S \) is called the left-representative (resp., right-representative) of \( S \).

Next, we wish to introduce the notion of contingency tree. Given any string \( x \), we conveniently set \( e_x = \lceil \log |x| \rceil \). Let \( C_t = (q, l_1, l_2, \sigma, r, t) \) denote an arbitrary marker of \( M \) on \( x \). To explain a contingency tree, we first define a contingency list at time \( t \), which consists of the sets described below. Let \( d \in \{0, e_x\} \mathbb{Z} \).

**Definition of Contingency Lists**

(a) \( L^{(0)}_{\text{last}}(d, l_2, t) \), which consists of all markers \( C \) satisfying the following requirement: there exist a \( d \)-section \( S \) and a \((d + 1)\)-section \( S' \) such that (i) \( C \) is the left-representative of \( S \), (ii) \( S \) is enclosed in \( S' \), and (iii) \( S' \) is the last-left-good section for \( C_t \).

(b) \( L^{(0)}_{\text{cur}}(d, l_2, t) \), which consists of all markers \( C \) satisfying the following requirement: there exist a \( d \)-section \( S \) and a \((d + 1)\)-section \( S' \) for which (i) \( C \) is the left-representative of \( S \), (ii) \( S \) is enclosed in \( S' \), (iii) \( S' \) is current for \( C_t \), and (iv) \( C \) is left-visible from \( C_t \).

(c) \( L^{(1)}_{\text{last}}(d, l_2, t) \) and \( L^{(1)}_{\text{cur}}(d, l_2, t) \), which are defined similarly to (a) and (b) by simply replacing ‘left’ with ‘right’.

Notice that, by the definition, the left-cut (resp., the right-cut) of \( C_t \) is the latest (i.e. the most recent) marker in \( L^{(0)}(0, l_2, t) \) (resp., \( L^{(0)}(0, l_2, t) \)). For every index \( d \in \{0, e_x\} \mathbb{Z} \) and any \( a \in \{l, r\} \), let \( L^{(a)}(d, l_2, t) = L^{(a)}_{\text{last}}(d, l_2, t) \cup L^{(a)}_{\text{cur}}(d, l_2, t) \) and set \( L(e_1, e_2, l_2, t) = \bigcup_{d \in \{0, e_x\} \mathbb{Z}} L^{(0)}(d, l_2, t) \bigcup \bigcup_{d \in \{0, e_x\} \mathbb{Z}} L^{(1)}(d, l_2, t) \) for any pair \( e_1, e_2 \in \{0, e_x\} \mathbb{Z} \). We always assume that all markers in \( L(e_1, e_2, l_2, t) \) are enumerated according to time. This contingency list \( L(e_1, e_2, l_2, t) \) is said to be linked to \( C_t \). A 2-contingency tree at time \( t \) is a two-node tree whose root is \( C_t \) and its child is a contingency list linked to \( C_t \). A \((k + 1)\)-contingency tree at time \( t \) is a levelled, rooted tree whose top 2 levels form a 2-contingency tree at time \( t \) and each marker (except for the root \( C_t \)) appearing in this 2-contingency tree is the root of another \( k \)-contingency tree at time \( t' \) with \( t' < t \). Figure 3 illustrates such a contingency tree. In particular, we treat \( C_0 \) as a special marker that links itself to a unique empty contingency list. To describe a \( k \)-contingency tree at time \( t \), we use the notation \( D(k, e_1, e_2, l_2, t) \). The principal contingency list of such a tree refers to the child of the root \( C_t \) of the tree.

**Lemma 5.4:** For any pair \( e_1, e_2 \in \{0, e_x\} \mathbb{Z} \), the total number of markers stored inside \( D(k, e_1, e_2, l_2, t) \) is \( O(\log^{k-2} n) \), where \( n = |x| \).
Figure 3. [left] A history of consecutive moves of a storage-tape head for \( k = 3 \). The leftmost three vertical dashed lines indicate \( d \)-sections for \( d = 3, 4, 5 \). The other vertical dashed lines indicate the tape cell numbers from 0 to 10 and the horizontal dotted lines show section time from 0 to 32. All storage-stationary-moves are suppressed into black circles and boxes. [right] A contingency tree, in which each node (except for the root) is a contingency list linked to a marker in the parent node.

Figure 4. A storage content history. Here, we assume that an underlying \( k \)-sda makes no storage-stationary-move. Thus, runtime matches section time.

**Proof:** Let \( n = |x| \). Each contingency list inside \( D(k, e_1, e_2, l_2, t) \) has only \( O(\log n) \) markers because \( 0 \leq e_1, e_2 \leq e_x = [\log n] \). Each node in \( D(k, e_1, e_2, l_2, t) \) except for the root is a contingency list and there are \( k-1 \) levels of such nodes in the tree. Since the second level of the tree has only one node, \( D(k, e_1, e_2, l_2, t) \) must contain \( O(\log^{k-2} n) \) markers.

### 5.2. Description of the main subroutine

We will describe in Section 5.3 how to simulate a polynomial-time depth-immune \( k \)-sda using only \( O(\log^k n) \) space. The core of this simulation procedure is the following subroutine, Subroutine \( A \), which manages \( k \)-contingency trees to recover the past behaviours of the depth-immune \( k \)-sda for determining its next moves.

**Subroutine \( A \):**

An input is of the form \((C_t, dv, e_1, e_2, D(dv, e_1, e_2, l_2, t)) \) with marker \( C_t = (q, l_1, l_2, \sigma, r, t) \in M_x, dv \in [0, k]_\mathbb{Z}, \) and \( e_1, e_2 \in [0, e_x]_\mathbb{Z} \). If \( dv \leq dv(C_{t+1}) \) holds for the \((t+1)\)-marker \( C_{t+1} = \)
(\nu, \ell', \ell_2, \sigma', \tau', t + 1), then \(A\) returns \((C_{t+1}, dv, e_1, e_2, \mathcal{D}(dv, e_1, e_2, \ell'_2, t + 1))\) by making a (possible) series of recursive calls to itself. Otherwise, it may return anything.

Hereafter, we explain how to implement Subroutine \(A\). We assume that all markers in each contingency list are automatically enumerated according to time. In this explanation, we use Figure 4 as a concrete example.

**[Implementation of Subroutine \(A\)]**

1. Consider the case where the storage-tape head makes a stationary move, that is, \(\ell'_2 = \ell_2\). Define \(r' = r\). Since we do not need to alter the contingency tree, we automatically set \(L^{(l)}(d, \ell'_2, t + 1) = L^{(l)}(d, \ell_2, t)\) and \(L^{(r)}(d', \ell'_2, t + 1) = L^{(r)}(d', \ell_2, t)\) for any \(d \in [0, e_1]_{\mathbb{Z}}\) and \(d' \in [0, e_2]_{\mathbb{Z}}\) in the principal contingency list of the contingency tree \(\mathcal{D}(dv, e_1, e_2, \ell'_2, t + 1)\). We further compute \(C_{t+1} = (q', \ell'_1, \ell'_2, \sigma', \tau', t + 1)\) from \(C_t\) directly by applying \(\delta\).

2. Consider the case where the storage-tape head is moving to the right, that is, \(\ell'_2 = \ell_2 + 1\). Define \(r' = r + 1\) and compute \(C_{t+1}\) and \(\mathcal{D}(dv, e_1, e_2, \ell'_2, t + 1)\) as follows.

   a. \([\text{list } L^{(l)}(d, \ell'_2, t + 1)\text{ for any } d \in [0, e_1]_{\mathbb{Z}}]\) Consider the principal contingency list \(L(e_1, e_2, \ell_2, t)\) of \(\mathcal{D}(dv, e_1, e_2, \ell_2, t)\). We compute \(L(e_1, e_2, \ell'_2, t + 1)\) in the following way. Take the maximum numbers \(d_0 \in [0, e_1]_{\mathbb{Z}}\) and \(d_2 \in [0, e_1]_{\mathbb{Z}}\) satisfying that \(r' \equiv 0 \pmod{2^{d_0}}\) and \(r \equiv 0 \pmod{2^{d_2}}\). For each index \(i \in [0, \max\{d_0, d_2\} - 1]_{\mathbb{Z}}\), we inductively define \(A_i\) by setting \(A_0 = L^{(l)}(0, \ell_2, t) \cup \{C_t\}\), and \(A_i = L^{(l)}(i - 1, \ell_2, t) \cup \{C_{i-1}\}\), where \(C_{i-1}\) is the oldest marker in \(A_{i-1}\).

   i. Assume that \(r'\) is even. Note that \(d_2 = 0\). For each \(i \in [0, d_0 - 1]_{\mathbb{Z}}\), we set \(L^{(l)}(i, \ell'_2, t + 1) = L^{(l)}(i, \ell_2, t + 1) = \emptyset, L^{(l)}(i, \ell', t + 1) = A_i, L^{(l)}(d_0, \ell'_2, t + 1) = A_{d_0},\) and \(L^{(l)}(d_0, \ell_2, t + 1) = L^{(l)}(d_0, \ell_2, t)\).

   ii. If \(r'\) is odd, then we set \(L^{(l)}(i, \ell'_2, t + 1) = A_i\) for any \(i \in [0, d_2]_{\mathbb{Z}}\). For all other indices \(d\) and all subscripts \(b \in \{\text{cur, last}\}\), we set \(L^{(l)}(d, \ell'_2, t + 1) = L^{(l)}(d, \ell_2, t)\) and \(L^{(l)}(d, \ell'_2, t + 1) = L^{(l)}(d, \ell_2, t)\). We also update the others without changing the content. In the example of \(t = 8\) in Figure 5, we have \(r' = 10\) and \(d_0 = 1\).

   iii. For all the remaining ‘nodes’ in the contingency tree \(\mathcal{D}(dv, e_1, e_2, \ell_2, t)\), we also update them in accordance with the above changes.

   b. \([\text{list } L^{(r)}(d', \ell'_2, t + 1)\text{ for any } d' \in [0, e_2]_{\mathbb{Z}}]\) This process is obtained straightforwardly from Item 3(a) by replacing ‘\(\ell\)’ with ‘\(r\)’, ‘left’ with ‘right’, ‘\(d\)’ with ‘\(d'\)’, and ‘\(e_1\)’ with ‘\(e_2\)’.

   c. \([\text{marker } C_{t+1}]\) To obtain the marker \(C_{t+1} = (q', \ell'_1, \ell'_2, \sigma', \tau', t + 1)\) from \(C_t\), it suffices to compute the left-cut (if any).

   i. Assume that \(dv \geq 0\). If there exists a left-cut \(C''\) of \(C_t\) at time \(< t\) either stored inside \(\mathcal{D}(dv, e_1, e_2, t)\) or obtained by Item 2(b), then the desired marker \(C_{t+1}\) is directly calculated from \(C_t\) and \(\text{symb}(C'')\) by applying \(\delta\). If there is no right-representative at location \(\geq \ell'_2\) at time \(< t\) inside \(\mathcal{D}(dv, e_1, e_2, t)\), then we know that the storage-tape head has never visited cell \(\ell'_2\), and thus cell \(\ell'_2\) must contain \(\emptyset\). By setting \(\sigma' = \emptyset\), we compute the \(\ell'_2\)-marker \(C_{t+1}\) from \(C_t\) alone by applying \(\delta\).

   ii. If \(dv < 0\), then cell \(\ell'_2\) must be already frozen. Let \(\sigma' = B\) and produce \(C_{t+1}\) from \(C_t\) alone by applying \(\delta\).

3. Let us consider the case where the storage-tape head is moving to the left, that is, \(\ell'_2 = \ell_2 - 1\). We set \(r' = r + 1\) and compute \(C_{t+1}\) and \(\mathcal{D}(dv, e_1, e_2, \ell'_2, t + 1)\) in the following way.

   a. \([\text{list } L^{(l)}(d, \ell'_2, t + 1)\text{ for any } d \in [0, e_1]_{\mathbb{Z}}]\) Consider the principal contingency list \(L(e_1, e_2, \ell_2, t)\) of \(\mathcal{D}(dv, e_1, e_2, \ell_2, t)\) and compute \(L(e_1, e_2, \ell'_2, t + 1)\) by taking the following procedure. Take the maximum number \(d_0 \in [0, e_1]_{\mathbb{Z}}\) satisfying \(r' \equiv 0 \pmod{2^{d_0}}\) and define
To be the minimal number in $[0, e_1]_\mathbb{Z}$ for which $|L^{(i)}(d_1, l_2, t)| \geq 2$ holds. In case that no such $d_1$ exists, we automatically set $d_1 = e_1$.

(i) For each index $d \in [0, d_1]_\mathbb{Z}$, we reset $L^{(i)}(d, l_2, t)$ to be $L^{(i)}(d, l_2, t) - \{C\}$, where $C$ is the newest marker in $L^{(i)}(d, l_2, t)$. The contingency tree $D(d, e_1, e_2, l_2, t)$ is then updated. If $d_1 = 0$, then we define $L^{(i)}(d, l_2, t + 1) = L^{(i)}(d, l_2, t)$ for any $d \in [0, e_1]_\mathbb{Z}$. In the example of $t = 18$ in Figure 6, we have $r' = 19$ and $d_0 = d_1 = 0$. Assume that $d_1 \neq 0$. If the left-cut of $C_{t+1}$ still remains in $D(d, e_1, e_2, l_2, t)$, then we set $C''$ to be this left-cut. This case is depicted as the example of $t = 15$ in Figure 6 with $r' = 16$, $d_0 = 4$, and $d_1 = 0$ and we obtain $C'' = C_{13}$. In the case where $D(d, e_1, e_2, l_2, t)$ contains no left-cut of $C_{t+1}$, we choose the latest marker $C'$ from $L^{(i)}(d_1, l_2, t)$ and define $l' = st-loc(C')$ and $t' = time(C')$. Since $d_1 > d_0$, $C'$ is the left-representative of the completed $d_1$-section. See the example of $t = 16$ in Figure 6 with $r' = 17$, $d_0 = d_1 = 2$, and $C' = C_8$. We then compute the left-cut $C''$ of $C_{t+1}$ that appears in a certain last-left-good $d'$-section with $d' < d_1$ satisfying both $st-loc(C'') = l'_2$ and $t' < time(C'') < t$. This is done by recursively calling $A$ as follows. Starting from $C'$ with $dv' = k$ and $D(dv', d_1 - 1, e_2, l', t')$, we inductively generate a pair of $\tilde{C}$ and $D(dv' - 1, d_1 - 1, e_2, l', t')$ with $\tilde{l} = time(\tilde{C})$ and $l = st-loc(\tilde{C})$, and run $A(\tilde{C}, dv' - 1, d_1 - 1, e_2, D(dv' - 1, d_1 - 1, e_2, l', t'))$ to obtain the next marker $\tilde{C}'$ and its contingency tree $D(dv' - 1, d_1 - 1, e_2, l', t + 1)$ with $\tilde{l}' = st-loc(\tilde{C}')$ until we reach $C''$ with $dv' \geq dv(C'')$. Note that the final recursive call must be of the form $A(\tilde{C}, dv - 1, d_1 - 1, e_2, D(dv - 1, d_1 - 1, e_2, l_2 - 1, t))$ with $l_2' = 1 = st-loc(\tilde{C})$. In the example of $t = 16$ in Figure 6, $C''$ is $C_{10}$.

(ii) We inductively define $B_0 = L^{(i)}(0, l_2', t') \cup \{C\}$ and $B_{i+1} = L^{(i)}(i, l_2', t') \cup \{\tilde{C}_i\}$ for any index $i \in [0, d_1 - 1]_\mathbb{Z}$, where $\tilde{C}_i$ is the oldest marker in $B_i$. We define $L^{(i)}(i, l_2', t + 1) = B_1$ and $L^{(i)}(i, l_2', t + 1) = \emptyset$ for all $i \in [0, d_1 - 1]_\mathbb{Z}$ and we then set $L^{(i)}(d_1, l_2', t + 1) = \emptyset$ and $L^{(i)}(d_1, l_2', t + 1) = L^{(i)}(d_1, l_2, t) \cup B_{d_1}$. In the example of $t = 16$ in Figure 6, we obtain $t' = 10$ because of $C'' = C_{10}$.

(iii) In the special case of $d_0 > d_1$, we additionally define $E_0 = \emptyset$ and $E_{i+1} = L^{(i)}(i, l_2, t) \cup \{\tilde{C}_i\}$ for any index $i \in [d_1 + 1, d_0 - 1]_\mathbb{Z}$, where $\tilde{C}_i$ is the oldest marker in $E_i$. We further set $L^{(i)}(i, l_2, t + 1) = E_i$, $L^{(i)}(i, l_2, t + 1) = \emptyset$ for any $i \in [d_1 + 1, d_0 - 1]_\mathbb{Z}$, and $L^{(i)}(d_0, l_2, t + 1) = E_{d_0}$. We then update all the others without changing the content of the old contingency lists $L^{(i)}(d, l_2, t)$. This process is exemplified in the case of $t = 15$ in Figure 6, where $r' = 16$, $d_0 = 4$, and $d_1 = 0$.

(iv) For all the other ‘nodes’ in $D(dv, e_1, e_2, l_2, t)$, we update them to match the aforementioned changes.

(2a) [list $L^{(i)}(d', l_2', t + 1)$ for any $d' \in [0, e_2]_\mathbb{Z}$] This process is obtained from Item 2(a) by replacing ‘t’ with ‘r’, ‘d’ with ‘dv’, and ‘e_1’ with ‘e_2’.

(c) [marker $C_{t+1}$] The desired $C_{t+1}$ is calculated in the following way.

(i) Assume that $dv \geq 1$. If there is a right-cut $C''$ of $C_t$ at time $< t$, which is either stored inside $D(dv, e_1, e_2, l_2, t)$ or obtained by Item 3(a), then the desired marker $C_{t+1}$ is directly calculated from $C_t$ and $symb(C'')$ by applying $\delta$. Assume otherwise. If there is a left-representative $C_{rep}$ for $C_t$ at time $0 < t$ and no left turn exists at location $l_2'$ at time $< t_0$, then we start with this left-representative $C_{rep}$ and compute a series of marker $\tilde{C}$ one by one (by incrementing time) from $C_{rep}$ using $\emptyset$ as inputs with no recursive call until we obtain the $l_2'$-marker $C'$. We compute the desired $l_2'$-marker $C_{t+1}$ from $C_t$ and $symb(C')$ by applying $\delta$.

(ii) If $dv \leq 0$, then cell $l_2'$ must be already frozen blank $B$. In this case, since we do not need the left-cut of $C_t$, we set $\sigma' = B$ and computer $C_{t+1}$ from $C_t$ along by applying $\delta$. 
5.3. Complexity analyses of the procedures

We have already given the description of Subroutine \( A \) in Section 5.2. What remains undone is to prove that the subroutine is correct and runs in polynomial time using only \( O(\log^k n) \) space.

In what follows, the depth of recursion with respect to inputs of length \( n \) refers to the maximum number of recursive calls invoked during any computation starting with any input of length \( n \).

Lemma 5.5: Let \( C_t \) and \( C_{t+1} \) be two markers at times \( t \) and \( t+1 \), respectively. Let \( e_1, e_2 \in [0, e_s]\) and let \( n \) denote the input length. If \( d > dv(C_{t+1}) \), then \( A(C_t, dv, e_1, e_2, D(dv, e_1, e_2, l_2, t)) \) correctly returns \((C_{t+1}, dv, e_1, e_2, D(dv, e_1, e_2, l_2', t + 1))\) and the depth of recursion is \( O(\log n) \).

Proof: We first argue the correctness of Subroutine \( A \). We assume that \( C_t = (q, l_1, l_2, \sigma, r, t) \) and \( D(dv, e_1, e_2, l_2, t) \) have already been correctly computed. We run \( A \) on an input of the form
(C_t, dv, e_1, e_2, D(dv, e_1, e_2, l_2, t)) with \( dc \geq dv(C_{t+1}) \). During the computation of \( A \), Items 2(b), 2(c), 3(a), and 3(c) require series of recursive calls to \( A \) itself. Notice that those recursive calls are possible because a contingency tree in, e.g. Item 3(a) contains \( D(dv - 1, d_1 - 1, e_2, t, t) \) linked to a left-representative \( C' \), which is included in \( D(dv, e_1, e_2, l_2, t) \). Let \( l'_2 \) be defined from \( l_2 \) by Items 1–3.

**Claim 1:** For any \( a \in \{l, r\} \), if \( L^{(a)}(d, l_2, t) \) is correct, then so is \( L^{(a)}(d, l'_2, t + 1) \).

**Proof:** Assume that \( L^{(a)}(d, l_2, t) \) is correctly computed. It suffices to check the following five items: 1, 2(a), 2(b), 3(a), and 3(b). Since Item 1 is obvious, we then consider Item 2(a), in which the tape head moves to the right. Recall the parameters used in the description of Item 2(a). If \( r' \neq \emptyset \), then a 1-section never ends at time \( t \). Note that \( d_1 \) equals the previous value of \( d_0 \) at time \( t-1 \). Thus, for any \( d' \leq d_2 \), we move all entries of \( 1 \)-section into \( (d + 1) \)-section. By contrast, if \( r' = \emptyset \), then every \( 1 \)-section ends at time \( t \) for any \( d \leq d_0 \). Since the current \( 0 \)-section changes, we make it empty at time \( t + 1 \) and move all entries in the current sections to their last-left-good sections by shifting the oldest markers in \( 1 \)-sections to \( (d + 1) \)-sections. Hence, \( L^{(1)}(d, l'_2, t + 1) \) is correctly computed.

Next, let us consider Item 3(a). If \( d_1 = 0 \), since the latest marker in \( L^{(1)}(d_1, l_2, t) \) is the left-cut, it should be deleted. We next assume that \( d_1 > 0 \). If the left-cut remains unchanged after the deletion of the newest marker, then we simply shift all entries from the current sections to their last-left-good sections. Assume otherwise. After updating \( L^{(1)}(d_1, l_2, t) \), the newest marker is the left-cut and is removed. A series of recursive calls helps us find the left-cut \( C''_t \) of \( C_{t+1} \). Up to \( d_1 = 1 \), we move all entries of the current sections to their last-left-good sections by first adding the left-cut \( C''_t \) and shifting the oldest markers in \( 1 \)-sections to \( (d + 1) \)-sections. In the case of \( d_0 > d_1 \), for all \( d \leq d_0 \), any \( 1 \)-section begins at section time \( t + 1 \). We move all entries in the last-left-good sections to the current sections by shifting the oldest markers in \( 1 \)-sections to \( (d + 1) \)-sections. Thus, \( L^{(1)}(d, l'_2, t + 1) \) contains the correct markers.

Items 2(b) and 3(b) are handled symmetrically to Items 3(a) and 2(a), respectively.

**Claim 2:** If \( L^{(a)}(d, l'_2, t + 1) \) and \( C_t \) are correctly computed for any \( a \in \{l, r\} \) and any \( i \in [0, t]\), then \( C_{t+1} \) is also correctly computed.

**Proof:** Assume that \( L^{(a)}(d, l'_2, t + 1) \) and \( C_t \) are correctly computed for any \( a \in \{l, r\} \) and any \( i \in [0, t]\). To verify the correctness of \( C_{t+1} \), it suffices to check three items: 1, 2(c), and 3(c). The case of Item 1 is obvious. Let us consider Item 2(c)(i). A left-cut \( C''_t \) of \( C_t \) at time \( < t \) contains the information on the previously written storage symbol at cell \( l'_2 \). Thus, we can easily obtain \( C_{t+1} \) correctly from this information together with \( C_t \) by applying \( \delta \). On the contrary, if there is no such left-cut, then we need only \( C_t \) to compute \( C_{t+1} \). In the case of Item 3(c)(i), if there is a right-cut of \( C_t \) at time \( < t \), then we can compute \( C_{t+1} \) correctly from this right-cut and \( C_t \). Otherwise, we can find an \( l'_2 \)-marker \( C' \) and then compute \( C_{t+1} \) directly. The remaining cases of Items 2(c)(ii) and 3(c)(ii) are obvious.

Next, we argue that the depth of recursion of \( A \) is \( O(\log n) \) by examining all the processes of \( A \) that invoke recursive calls. Let us first consider Item 3 of \( A \). To compute \( A(\tilde{C}, dv', e_1, l_1, t') \) in Item 3(a), if there is no right-cut \( C''_t \) for \( C_t \), then we need to make a series of recursive calls of the form \( A(\tilde{C}, dv', e_1, l_1, t') \). Note that each recursive call depends on the parameter values of \( dv' \) and \( d_1 \). Note that \( d_1 \) is smaller than \( e_1 \). To compute \( C_{t+1} \) with \( l'_2 = st-loc(C_{t+1}) \) in Item 3(c), we need to compute \( m \) markers \( C^{(1)}, C^{(2)}, \ldots, C^{(m)} \) for a certain number \( m \in \mathbb{N}^+ \) with \( l'_2 = st-loc(C^{(l)}) \) and \( 0 \leq time(C^{(i+1)}) < time(C^{(i)}) < t \) for any \( i \in [m] \). If \( m > dv(C_{t+1}) \), then \( A \) eventually acknowledges that cell \( l'_2 \) has already become frozen blank and it correctly computes \( C_{t+1} \). At that point, the recursion ends. Thus, \( m \) must be at most \( k \).

A similar argument is applicable to Items 2(a) and 2(c). Overall, since \( e_1, e_2 \in [0, e_x] \) and \( e_x = \lceil \log |x| \rceil \), the depth of recursion must be \( O(\log n) \).
Finally, we prove Theorem 5.1, which provides an SC$^k$-upper bound on the computational complexity of kSDA$^{\text{imm}}$ for each index $k \geq 2$.

**Proof:** Take an arbitrary language $L$ in kSDA$^{\text{imm}}$ and any depth-immune k-sda $M$ that recognizes $L$ in polynomial time. Consider the following procedure that simulates $M$ step by step on input $x$. Initially, we prepare the initial marker $C_0 = (q_0, 0, 0, 0, 0)$ and the unique $k$-contingency list $D(k, e_x, e_x, 0, 0)$ linked to $C_0$.

**Simulation Procedure $P$ for $M$:**

On input $x \in \Sigma^*$, set $n = |x|$ and $e_x = \lceil \log |x| \rceil$, prepare $C_0$ and $D(k, e_x, e_x, 0, 0)$. Set $I_0 = (C_0, k, e_x, e_x, D(k, e_x, e_x, 0, 0))$. Recursively, we run Subroutine $A$ on $I_i$ and produce $I_{i+1}$ until $M$ halts at time, say, $t$ with $t \leq p(n)$. Let $I_t = (C_t, k, e_x, e_x, D(k, e_x, e_x, t, t))$ denote the final outcome of $A$. If $C_t$ contains an accepting state, then we accept $x$; otherwise, we reject $x$.

By Lemma 5.5, if $I_i$ correctly represents the configuration of $M$ at time $i$, then Subroutine $A$ correctly computes $I_{i+1}$. By the mathematical induction, we conclude that $I_t$ correctly represents the halting configuration of $M$. This implies that the simulation procedure $P$ correctly determines whether or not $x$ belongs to $L$.

To store a contingency tree requires $O(\log^{k-1} n)$ bits by Lemma 5.4 since each contingency list in the tree needs $O(\log n)$ bits to express. Lemma 5.5 then concludes that we need only polynomial runtime and $O(\log^k n)$ memory bits to perform the simulation procedure $P$. Therefore, $L$ belongs to SC$^k$. 

**6. A brief discussion and future directions**

As access-controlled extensions of deterministic pushdown automata, we have introduced a new machine model of one-way deterministic depth-$k$ storage automata (or k-sda’s). We have then studied the fundamental properties of languages recognized by such machines and those of languages that are logarithmic-space many-one reducible (or L-m-reducible) to them. We have called by kSDA resp., kSDA$^{\text{imm}}$ the language family induced by depth-susceptible k-sda’s (resp., depth-immune k-sda’s). This machine model naturally expands Hibbard’s model of scan limited automata [12]. The L-m-closure of kSDA (resp., kSDA$^{\text{imm}}$) is succinctly denoted by LOGkSDA (resp., LOGkSDA$^{\text{imm}}$). In Section 3, we have presented two machine characterizations of LOGkSDA. In Section 4, we have also constructed a generic L-m-hard problem for LOGkSDA. In Section 5, we have shown that kSDA$^{\text{imm}} \subseteq$ SC$^k$ for each $k \geq 2$.

As future research directions, we list six important questions that have been neither discussed nor solved in this exposition.

1. [class separations] As for Hibbard’s k-lda’s, it is known that, for every $k \in \mathbb{N}^+$, $(k+1)$-lda’s are in general more powerful in computational power than $k$-lda’s [12]. Due to the similarity between k-lda’s and k-sda’s regarding the use of ‘access-controlled’ memory devices, we can anticipate that, in general, $(k+1)$-sda’s may have more computational power than k-sda’s. It is thus imperative to prove the difference between kSDA and $(k+1)$SDA for any index $k \geq 2$.

2. [better upper bounds] We have shown in Section 5 that kSDA$^{\text{imm}} \subseteq$ SC$^k$ for any index $k \geq 2$. However, we do not know whether SC$^k$ is the best possible upper bound that we can achieve. We may ask whether or not the computational complexity of kSDA$^{\text{imm}}$ is much lower than SC$^k$. For instance, is it true that kSDA$^{\text{imm}} \subseteq$ SC$^{k-1}$?

3. [nondeterminism and randomization] In this exposition, we have defined only the deterministic storage automata. Other important variants include a nondeterministic analogue of $k$-sda’s, called nondeterministic depth-$k$ storage automata (or k-sna’s), and a randomized (or probabilistic) analogue of $k$-sda’s, called probabilistic depth-$k$ storage automata (or k-spa’s). If we write
kSNA and kSBPA to respectively denote the family of all languages recognized by k-sna’s and that of all languages recognized by bounded-error k-spa’s, then obviously kSNA is a subclass of NP and kSBPA is a subclass of BPP. It is important to study the basic properties of these language families and relationships between them.

(4) [structural properties] Another important subject is (structural) closure properties under language operations (such as union, intersection, and concatenation). Since we have not discussed in this exposition any closure property of kSDA (as well as kSDAimm), we expect that a study on the closure properties may reveal fruitful features of kSDA.

(5) [depth-immune model] As noted in Section 1, LOGDCFL is known to be characterized by numerous models, even with no use of L-m-reductions. Similarly, we have given in Section 3 two machine characterizations of LOGkSDA. Is there any ‘natural’ model that can capture the depth-immune counterpart, LOGkSDAimm, with no use of L-m-reductions?

(6) [natural hard problems] In Section 4, for each index \( k \geq 2 \), we have constructed a language, MEMB\(_k\), which is hard for LOGkSDA under L-m-reductions. This language MEMB\(_k\) is generic but looks quite artificial. It is therefore desirable to find ‘natural’ hard (or even complete) problems for LOGkSDA.

Notes

1. A read-only tape is called read once if, whenever it reads a tape symbol (except for \( \varepsilon \)-moves, if any), it must move to the next unread cell.
2. This claim comes from the fact that Hibbard’s rewriting systems can be forced to satisfy the blank-skipping property without compromising their computational power [25].
3. The use of stationary move is made in this exposition only for convenience sake. It is also possible to define k-sda’s using \( \varepsilon \)-moves in place of stationary moves.

Disclosure statement

No potential conflict of interest was reported by the author(s).

References

[1] B. Braunmühl, S. Cook, K. Mehlhorn, and R. Verbeek, The recognition of deterministic CFLs in small time and space, Inf. Control 56 (1983), pp. 34–51.
[2] T. Chan, Pushdown automata with reversal-bounded counters, J. Comput. Syst. Sci. 37 (1988), pp. 269–291.
[3] S.A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, J. ACM 18 (1971), pp. 4–18.
[4] S.A. Cook, Deterministic CFLs are accepted simultaneously in polynomial time and log squared space, in The Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC’79), 1979, pp. 338–345.
[5] P. Dymond and W. Ruzzo, Parallel RAMs with owned global memory and deterministic context-free language recognition, in The Proceedings of the 13th International Colloquium on Automata, Languages and Programming (ICALP’86), Lecture Notes in Computer Science Vol. 226, Springer, 1986, pp. 95–104.
[6] H. Fernau, K.J. Lange, and K. Reinhardt, Advocating owership, in The Proceedings of the 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FST&FTCS’96), Lecture Notes in Computer Science Vol. 1180, Springer, 1996, pp.286–297.
[7] Z. Galil, Some open problems in the theory of computation as questions about two-way deterministic pushdown automaton languages, Math. Syst. Theory 10 (1977), pp. 211–228.
[8] S. Ginsburg and S. Greibach, Deterministic context free languages, Inf. Control 9 (1966), pp. 620–648.
[9] S. Ginsburg, S.A. Greibach, and M.A. Harrison, Stack automata and compiling, J. ACM 14 (1967), pp. 172–201.
[10] S. Ginsburg, S.A. Greibach, and M.A. Harrison, One-way stack automata, J. ACM 14 (1967), pp. 389–418.
[11] J. Hartmanis, On non-determinacy in simple computing devices, Acta Inf. 1 (1972), pp. 336–344.
[12] T.N. Hibbard, A generalization of context-free determinism, Inf. Control 11 (1967), pp. 196–238.
[13] O.H. Ibarra, Characterizations of some tapes and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata, J. Comput. Syst. Sci. 5 (1971), pp. 88–117.
[14] M. Kutrib, G. Pighizzini, and M. Wendlandt, Descriptional complexity of limited automata, Inf. Comput. 259 (2018), pp. 259–276.
[15] M. Kutrib and M. Wendlandt, Reversible limited automata, Fund. Inf. 155 (2017), pp. 31–58.
[16] L.Y. Liu and P. Weiner, *An infinite hierarchy of intersections of context-free languages*, Math. Syst. Theory 7 (1973), pp. 185–192.

[17] M. Lohrey, *Decidability and complexity in automatic monoids*, Int. Found. Comput. Sci. 16 (2005), pp. 707–722.

[18] P. McKenzie, K. Reinhardt, and V. Vinay, *Circuits and context-free languages*, in *The Proceedings of the 5th Annual International Computing and Combinatorics Conference (COCOON’99)*, Lecture Notes in Computer Science, Springer, vol. 1627, 1999, pp. 194–203.

[19] A. Meduna, *Deep pushdown automata*, Acta Inf. 42 (2006), pp. 541–552.

[20] G. Pighizzini and A. Pisoni, *Limited automata and regular languages*, Int. J. Found. Comput. Sci. 25 (2014), pp. 897–916.

[21] G. Pighizzini and A. Pisoni, *Limited automata and context-free languages*, Fund. Inf. 136 (2015), pp. 157–176.

[22] G. Pighizzini and L. Prigioniero, *Limited automata and unary languages*, Inf. Comput. 266 (2019), pp. 60–74.

[23] I.H. Sudborough, *On the tape complexity of deterministic context-free languages*, J. ACM 25 (1978), pp. 405–414.

[24] K. Tadaki, T. Yamakami, and J.C.H. Lin, *Theory of one-tape linear-time Turing machines* Theor. Comput. Sci. 411 (2010), pp. 22–43. An extended abstract appeared in the Proc. of the 30th SOFSEM Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2004), Lecture Notes in Computer Science, Springer, vol. 2932, 2004, pp. 335–348.

[25] T. Yamakami, Behavioral strengths and weaknesses of various models of limited automata, in *The Proceeding of the 45th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2019)*, Lecture Notes in Computer Science Vol. 11376, Springer, 2019, pp. 519–530. A complete and corrected version is available at arXiv:2111.05000, 2021.

[26] T. Yamakami, Intersection and union hierarchies of deterministic context-free languages and pumping lemmas, in *The Proceedings of the 15th International Conference on Language and Automata Theory and Applications (LATA 2020)*, Lecture Notes in Computer Science Vol. 12038, Springer, 2020, pp. 341–353. A complete and corrected version is available at arXiv:2112.09383, 2021.