Sublinear-Time Language Recognition and Decision by One-Dimensional Cellular Automata

Augusto Modanese(✉)

Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
modanese@kit.edu

Abstract. After an apparent hiatus of roughly 30 years, we revisit a seemingly neglected subject in the theory of (one-dimensional) cellular automata: sublinear-time computation. The model considered is that of ACAs, which are language acceptors whose acceptance condition depends on the states of all cells in the automaton. We prove a time hierarchy theorem for sublinear-time ACA classes, analyze their intersection with the regular languages, and, finally, establish strict inclusions in the parallel computation classes SC and (uniform) AC. As an addendum, we introduce and investigate the concept of a decider ACA (DACA) as a candidate for a decider counterpart to (acceptor) ACAs. We show the class of languages decidable in constant time by DACAs equals the locally testable languages, and we also determine $\Omega(\sqrt{n})$ as the (tight) time complexity threshold for DACAs up to which no advantage compared to constant time is possible.

1 Introduction

While there have been several works on linear- and real-time language recognition by cellular automata over the years (see, e.g., [14,24] for an overview), interest in the sublinear-time case has been scanty at best. We can only speculate this has been due to a certain obstinacy concerning what is now the established acceptance condition for cellular automata, namely that the first cell determines the automaton's response, despite alternatives being long known [18]. Under this condition, only a constant-size prefix can ever influence the automaton's decision, which effectively dooms sublinear time to be but a trivial case just as it is for (classical) Turing machines, for example. Nevertheless, at least in the realm of Turing machines, this shortcoming was readily circumvented by adding a random access mechanism to the model, thus sparking rich theories on parallel computation [5,20], probabilistically checkable proofs [23], and property testing [8,19].

In the case of cellular automata, the adaptation needed is an alternate (and by all means novel) acceptance condition, covered in Sect. 2. Interestingly, in the

Some proofs have been omitted due to page constraints. These may be found in the full version of the paper [17].

N. Jonoska and D. Savchuk (Eds.): DLT 2020, LNCS 12086, pp. 251–265, 2020.
https://doi.org/10.1007/978-3-030-48516-0_19

resulting model, called ACA, parallelism and local behavior seem to be more marked features, taking priority over cell communication and synchronization algorithms (which are the dominant themes in the linear- and real-time constructions). As mentioned above, the body of theory on sublinear-time ACAs is very small and, to the best of our knowledge, resumes itself to [10,13,21]. Ibarra et al. [10] show sublinear-time ACAs are capable of recognizing non-regular languages and also determine a threshold (namely $\Omega(\log n)$) up to which no advantage compared to constant time is possible. Meanwhile, Kim and McCloskey [13] and Sommerhalder and Westrhenen [21] analyze the constant-time case subject to different acceptance conditions and characterize it based on the locally testable languages, a subclass of the regular languages.

Indeed, as covered in Sect. 3, the defining property of the locally testable languages, that is, that words which locally appear to be the same are equivalent with respect to membership in the language at hand, effectively translates into an inherent property of acceptance by sublinear-time ACAs. In Sect. 4, we prove a time hierarchy theorem for sublinear-time ACAs as well as further relate the language classes they define to the regular languages and the parallel computation classes SC and (uniform) AC. In the same section, we also obtain an improvement on a result of [10]. Finally, in Sect. 5 we consider a plausible model of ACAs as language deciders, that is, machines which must not only accept words in the target language but also explicitly reject those which do not. Section 6 concludes.

2 Definitions

We assume the reader is familiar with the theory of formal languages and cellular automata as well as with computational complexity theory (see, e.g., standard references [1,6]). This section reviews basic concepts and introduces ACAs.

$\mathbb{Z}$ denotes the set of integers, $\mathbb{N}_+$ that of (strictly) positive integers, and $\mathbb{N}_0 = \mathbb{N}_+ \cup \{0\}$. B^A is the set of functions $f\colon A \to B$. For a word $w \in \Sigma^*$ over an alphabet Σ, $w(i)$ is the i-th symbol of w (starting with the 0-th symbol), and $|w|_x$ is the number of occurrences of $x \in \Sigma$ in w. For $k \in \mathbb{N}_0$, $p_k(w)$, $s_k(w)$, and $I_k(w)$ are the prefix, suffix and set of infixes of length k of w, respectively, where $p_{k'}(w) = s_{k'}(w) = w$ and $I_{k'}(w) = \{w\}$ for $k' \geq |w|$. $\Sigma^{\leq k}$ is the set of words $w \in \Sigma^*$ for which $|w| \leq k$. Unless otherwise noted, n is the input length.

2.1 (Strictly) Locally Testable Languages

The class REG of regular languages is defined in terms of (deterministic) automata with finite memory and which read their input in a single direction (i.e., from left to right), one symbol at a time; once all symbols have been read, the machine outputs a single bit representing its decision. In contrast, a *scanner* is a memoryless machine which reads a span of $k \in \mathbb{N}_+$ symbols at a time of an input provided with start and end markers (so it can handle prefixes and suffixes separately); the scanner validates every such substring it reads using the same

predicate, and it accepts if and only if all these validations are successful. The languages accepted by these machines are the strictly locally testable languages.[1]

Definition 1 (strictly locally testable). *Let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is* strictly locally testable *if there is some $k \in \mathbb{N}_+$ and sets $\pi, \sigma \subseteq \Sigma^{\leq k}$ and $\mu \subseteq \Sigma^k$ such that, for every word $w \in \Sigma^*$, $w \in L$ if and only if $p_k(w) \in \pi$, $I_k(w) \subseteq \mu$, and $s_k(w) \in \sigma$. SLT is the class of strictly locally testable languages.*

A more general notion of locality is provided by the locally testable languages. Intuitively, L is locally testable if a word w being in L or not is entirely dependent on a property of the substrings of w of some constant length $k \in \mathbb{N}_+$ (that depends only on L, not on w). Thus, if any two words have the same set of substrings of length k, then they are equivalent with respect to being in L:

Definition 2 (locally testable). *Let Σ be an alphabet. A language $L \subseteq \Sigma^*$ is* locally testable *if there is some $k \in \mathbb{N}_+$ such that, for every $w_1, w_2 \in \Sigma^*$ with $p_k(w_1) = p_k(w_2)$, $I_k(w_1) = I_k(w_2)$, and $s_k(w_1) = s_k(w_2)$ we have that $w_1 \in L$ if and only if $w_2 \in L$. LT denotes the class of locally testable languages.*

LT is the *Boolean closure* of SLT, that is, its closure under union, intersection, and complement [16]. In particular, $\mathsf{SLT} \subsetneq \mathsf{LT}$ (i.e., the inclusion is proper [15]).

2.2 Cellular Automata

In this paper, we are strictly interested in one-dimensional cellular automata with the standard neighborhood. For $r \in \mathbb{N}_0$, let $N_r(z) = \{z' \in \mathbb{Z} \mid |z - z'| \leq r\}$ denote the *extended neighborhood of radius r* of the cell $z \in \mathbb{Z}$.

Definition 3 (cellular automaton). *A* cellular automaton *(*CA*) C is a triple (Q, δ, Σ) where Q is a finite, non-empty set of* states*, $\delta\colon Q^3 \to Q$ is the* local transition function*, and $\Sigma \subseteq Q$ is the* input alphabet*. An element of Q^3 (resp., $Q^{\mathbb{Z}}$) is called a* local *(resp.,* global*)* configuration *of C. δ induces the* global transition function *$\Delta\colon Q^{\mathbb{Z}} \to Q^{\mathbb{Z}}$ on the configuration space $Q^{\mathbb{Z}}$ by $\Delta(c)(z) = \delta(c(z-1), c(z), c(z+1))$, where $z \in \mathbb{Z}$ is a cell and $c \in Q^{\mathbb{Z}}$.*

Our interest in CAs is as machines which receive an input and process it until a final state is reached. The input is provided from left to right, with one cell for each input symbol. The surrounding cells are inactive and remain so for the entirety of the computation (i.e., the CA is bounded). It is customary for CAs to have a distinguished cell, usually cell zero, which communicates the machine's output. As mentioned in the introduction, this convention is inadequate for computation in sublinear time; instead, we require the finality condition to depend on the entire (global) configuration (modulo inactive cells):

[1] The term "(locally) testable in the strict sense" ((L)TSS) is also common [13,15,16].

Fig. 1. Computation of an ACA which recognizes $L = \{01\}^+$. The input words are $010101 \in L$ and $001010 \notin L$, respectively.

Definition 4 (CA computation). *There is a distinguished state $q \in Q \setminus \Sigma$, called the* inactive state*, which, for every $z_1, z_2, z_3 \in Q$, satisfies $\delta(z_1, z_2, z_3) = q$ if and only if $z_2 = q$. A cell not in state q is said to be* active*. For an input $w \in \Sigma^*$, the* initial configuration *$c_0 = c_0(w) \in Q^{\mathbb{Z}}$ of C for w is $c_0(i) = w(i)$ for $i \in \{0, \dots, |w| - 1\}$ and $c_0(i) = q$ otherwise. For $F \subseteq Q \setminus \{q\}$, a configuration $c \in Q^{\mathbb{Z}}$ is F-*final *(for w) if there is a (minimal) $\tau \in \mathbb{N}_0$ such that $c = \Delta^\tau(c_0)$ and c contains only states in $F \cup \{q\}$. In this context, the sequence $c_0, \dots, \Delta^\tau(c_0) = c$ is the* trace *of w, and τ is the* time complexity *of C (with respect to F and w).*

Because we effectively consider only bounded CAs, the computation of w involves exactly $|w|$ active cells. The surrounding inactive cells are needed only as markers for the start and end of w. As a side effect, the initial configuration $c_0 = c_0(\varepsilon)$ for the empty word ε is stationary (i.e., $\Delta(c_0) = c_0$) regardless of the choice of δ. Since this is the case only for ε, we disregard it for the rest of the paper, that is, we assume it is not contained in any of the languages considered.

Finally, we relate final configurations and computation results. We adopt an acceptance condition as in [18,21] and obtain a so-called ACA; here, the "A" of "ACA" refers to the property that all (active) cells are relevant for acceptance.

Definition 5 (ACA). *An* ACA *is a CA C with a non-empty subset $A \subseteq Q \setminus \{q\}$ of* accept states*. For $w \in \Sigma^+$, if C reaches an A-final configuration, we say $C* accepts *$w$. $L(C)$ denotes the set of words accepted by C. For $t\colon \mathbb{N}_+ \to \mathbb{N}_0$, we write $\mathsf{ACA}(t)$ for the class of languages accepted by an ACA with time complexity bounded by t, that is, for which the time complexity of accepting w is $\leq t(|w|)$.*

$\mathsf{ACA}(t_1) \subseteq \mathsf{ACA}(t_2)$ is immediate for functions $t_1, t_2\colon \mathbb{N}_+ \to \mathbb{N}_0$ with $t_1(n) \leq t_2(n)$ for every $n \in \mathbb{N}_+$. Because Definition 5 allows multiple accept states, it is possible for each (non-accepting) state z to have a corresponding accept state z_A. In the rest of this paper, when we say a cell becomes (or marks itself as) accepting (without explicitly mentioning its state), we intend to say it changes from such a state z to z_A.

Figure 1 illustrates the computation of an ACA with input alphabet $\Sigma = \{0, 1\}$ and which accepts $\{01\}^+$ with time complexity equal to one (step). The local transition function is such that $\delta(0, 1, 0) = \delta(1, 0, 1) = \delta(q, 0, 1) = \delta(0, 1, q) = a$, a being the (only) accept state, and $\delta(z_1, z_2, z_3) = z_2$ for $z_2 \neq a$ and arbitrary z_1 and z_3.

3 First Observations

This section recalls results on sublinear-time ACA computation (i.e., $\mathsf{ACA}(t)$ where $t \in o(n)$) from [10,13,21] and provides some additional remarks. We start

with the constant-time case (i.e., $\mathsf{ACA}(O(1))$). Here, the connection between scanners and ACAs is apparent: If an ACA accepts an input w in time $\tau = \tau(w)$, then w can be verified by a scanner with an input span of $2\tau + 1$ symbols and using the predicate induced by the local transition function of the ACA (i.e., the predicate is true if and only if the symbols read correspond to $N_\tau(z)$ for some cell z in the initial configuration and z is accepting after τ steps).

Constant-time ACA computation has been studied in [13,21]. Although in [13] we find a characterization based on a hierarchy over SLT, the acceptance condition there differs slightly from that in Definition 5; in particular, the automata there run for a number of steps which is fixed for each automaton, and the outcome is evaluated (only) in the final step. In contrast, in [21] we find the following, where $\mathsf{SLT}_\vee$ denotes the closure of SLT under union:

Theorem 6 ([21]). $\mathsf{ACA}(O(1)) = \mathsf{SLT}_\vee$.

Thus, $\mathsf{ACA}(O(1))$ is closed under union. In fact, more generally:

Proposition 7. *For any $t\colon \mathbb{N}_+ \to \mathbb{N}_+$, $\mathsf{ACA}(O(t))$ is closed under union.*

$\mathsf{ACA}(O(1))$ is closed under intersection [21]. It is an open question whether $\mathsf{ACA}(O(t))$ is also closed under intersection for every $t \in o(n)$.

Moving beyond constant time, in [10] we find the following:

Theorem 8 ([10]). *For $t \in o(\log n)$, $\mathsf{ACA}(t) \subseteq \mathsf{REG}$.*

In [10] we find an example for a non-regular language in $\mathsf{ACA}(O(\log n))$ which is essentially a variation of the language

$$\mathsf{BIN} = \{\mathrm{bin}_k(0)\#\mathrm{bin}_k(1)\#\cdots\#\mathrm{bin}_k(2^k-1) \mid k \in \mathbb{N}_+\}$$

where $\mathrm{bin}_k(m)$ is the k-digit binary representation of $m \in \{0,\ldots,2^k-1\}$.

To illustrate the ideas involved, we present an example related to BIN (though it results in a different time complexity) and which is also useful in later discussions in Sect. 5. Let $w_k(i) = 0^i 10^{k-i-1}$ and consider the language

$$\mathsf{IDMAT} = \{w_k(0)\#w_k(1)\#\cdots\#w_k(k-1) \mid k \in \mathbb{N}_+\}$$

of all identity matrices in line-for-line representations, where the lines are separated by # symbols.[2]

We now describe an ACA for IDMAT; the construction closely follows the aforementioned one for BIN found in [10] (and the difference in complexity is only due to the different number and size of blocks in the words of IDMAT and BIN). Denote each group of cells initially containing a (maximally long) $\{0,1\}^+$ substring of $w \in \mathsf{IDMAT}$ by a *block*. Each block of size b propagates its contents to the neighboring blocks (in separate registers); using a textbook CA technique, this requires exactly $2b$ steps. Once the strings align, a block initially

[2] Alternatively, one can also think of IDMAT as a (natural) problem on graphs presented in the adjacency matrix representation.

containing $w_k(i)$ verifies it has received $w_k(i-1)$ and $w_k(i+1)$ from its left and right neighbor blocks (if either exists), respectively. The cells of a block and its delimiters become accepting if and only if the comparisons are successful and there is a single # between the block and its neighbors. This process takes linear time in b; since any $w \in \mathsf{IDMAT}$ has $O(\sqrt{|w|})$ many blocks, each with $b \in O(\sqrt{|w|})$ cells, it follows that $\mathsf{IDMAT} \in \mathsf{ACA}(O(\sqrt{n}))$.

To show the above construction is time-optimal, we use the following observation, which is also central in proving several other results in this paper:

Lemma 9. *Let C be an ACA, and let w be an input which C accepts in (exactly) $\tau = \tau(w)$ steps. Then, for every input w' such that $p_{2\tau}(w) = p_{2\tau}(w')$, $I_{2\tau+1}(w') \subseteq I_{2\tau+1}(w)$, and $s_{2\tau}(w) = s_{2\tau}(w')$, C accepts w' in at most τ steps.*

The lemma is intended to be used with $\tau < \frac{|w|}{2}$ since otherwise $w = w'$. It can be used, for instance, to show that $\mathsf{SOMEONE} = \{w \in \{0,1\}^+ \mid |w|_1 \geq 1\}$ is not in $\mathsf{ACA}(t)$ for any $t \in o(n)$ (e.g., set $w = 0^k 10^k$ and $w' = 0^{2k+1}$ for large $k \in \mathbb{N}_+$). It follows $\mathsf{REG} \not\subseteq \mathsf{ACA}(t)$ for $t \in o(n)$.

Since the complement of $\mathsf{SOMEONE}$ (respective to $\{0,1\}^+$) is $\{0\}^+$ and $\{0\}^+ \in \mathsf{ACA}(O(1))$ (e.g., simply set 0 as the ACA's accepting state), $\mathsf{ACA}(t)$ is not closed under complement for any $t \in o(n)$. Also, $\mathsf{SOMEONE}$ is a regular language and $\mathsf{BIN} \in \mathsf{ACA}(O(\log n))$ is not, so we have:

Proposition 10. *For $t \in \Omega(\log n) \cap o(n)$, $\mathsf{ACA}(t)$ and REG are incomparable.*

If the inclusion of infixes in Lemma 9 is strengthened to an equality, one may apply it in both directions and obtain the following stronger statement:

Lemma 11. *Let C be an ACA with time complexity bounded by $t\colon \mathbb{N}_+ \to \mathbb{N}_0$ (i.e., C accepts any input of length n in at most $t(n)$ steps). Then, for any two inputs w and w' with $p_{2\mu}(w) = p_{2\mu}(w')$, $I_{2\mu+1}(w) = I_{2\mu+1}(w')$, and $s_{2\mu}(w) = s_{2\mu}(w')$ where $\mu = \max\{t(|w|), t(|w'|)\}$, we have that $w \in L(C)$ if and only if $w' \in L(C)$.*

Finally, we can show our ACA for IDMAT is time-optimal:

Proposition 12. *For any $t \in o(\sqrt{n})$, $\mathsf{IDMAT} \notin \mathsf{ACA}(t)$.*

4 Main Results

In this section, we present various results regarding $\mathsf{ACA}(t)$ where $t \in o(n)$. First, we obtain a time hierarchy theorem, that is, under plausible conditions, $\mathsf{ACA}(t') \subsetneq \mathsf{ACA}(t)$ for $t' \in o(t)$. Next, we show $\mathsf{ACA}(t) \cap \mathsf{REG}$ is (strictly) contained in LT and also present an improvement to Theorem 8. Finally, we study inclusion relations between $\mathsf{ACA}(t)$ and the SC and (uniform) AC hierarchies. Save for the material covered so far, all three subsections stand out independently from one another.

4.1 Time Hierarchy

For functions $f, t\colon \mathbb{N}_+ \to \mathbb{N}_0$, we say f is *time-constructible by CAs in $t(n)$ time* if there is a CA C which, on input 1^n, reaches a configuration containing the value $f(n)$ (binary-encoded) in at most $t(n)$ steps.[3] Note that, since CAs can simulate (one-tape) Turing machines in real-time, any function constructible by Turing machines (in the corresponding sense) is also constructible by CAs.

Theorem 13. *Let $f \in \omega(n)$ with $f(n) \leq 2^n$, $g(n) = 2^{n-\lfloor \log f(n) \rfloor}$, and let f and g be time-constructible (by CAs) in $f(n)$ time. Furthermore, let $t\colon \mathbb{N}_+ \to \mathbb{N}_0$ be such that $3f(k) \leq t(f(k)g(k)) \leq cf(k)$ for some constant $c \geq 3$ and all but finitely many $k \in \mathbb{N}_+$. Then, for every $t' \in o(t)$, $\mathsf{ACA}(t') \subsetneq \mathsf{ACA}(t)$.*

Given $a > 1$, this can be used, for instance, with any time-constructible $f \in \Theta(n^a)$ (resp., $f \in \Theta(2^{n/a})$, in which case $a = 1$ is also possible) and $t \in \Theta((\log n)^a)$ (resp., $t \in \Theta(n^{1/a})$). The proof idea is to construct a language L similar to BIN (see Sect. 3) in which every $w \in L$ has length exponential in the size of its blocks while the distance between any two blocks is $\Theta(t(|w|))$. Due to Lemma 9, the latter implies L is not recognizable in $o(t(|w|))$ time.

Proof. For simplicity, let $f(n) > n$. Consider $L = \{w_k \mid k \in \mathbb{N}_+\}$ where

$$w_k = \mathrm{bin}_k(0)\#^{f(k)-k}\mathrm{bin}_k(1)\#^{f(k)-k} \cdots \mathrm{bin}_k(g(k)-1)\#^{f(k)-k}$$

and note $|w_k| = f(k)g(k)$. Because $t(|w_k|) \in O(f(k))$ and $f(k) \in \omega(k)$, given any $t' \in o(t)$, setting $w = w_k$, $w' = 0^k\#^{|w_k|-k}$, and $\tau = t'(|w_k|)$ and applying Lemma 9 for sufficiently large k yields $L \notin \mathsf{ACA}(t')$.

By assumption it suffices to show $w = w_k \in L$ is accepted by an ACA C in at most $3f(k) \leq t(|w|)$ steps for sufficiently large $k \in \mathbb{N}_+$. The cells of C perform two procedures P_1 and P_2 simultaneously: P_1 is as in the ACA for BIN (see Sect. 3) and ensures that the blocks of w have the same length, that the respective binary encodings are valid, and that the last value is correct (i.e., equal to $g(k) - 1$). In P_2, each block computes $f(k)$ as a function of its block length k. Subsequently, the value $f(k)$ is decreased using a real-time counter (see, e.g., [12] for a construction). Every time the counter is decremented, a signal starts from the block's leftmost cell and is propagated to the right. This allows every group of cells of the form bs with $b \in \{0,1\}^+$ and $s \in \{\#\}^+$ to assert there are precisely $f(k)$ symbols in total (i.e., $|bs| = f(k)$). A cell is accepting if and only if it is accepting both in P_1 and P_2. The proof is complete by noticing either procedure takes a maximum of $3f(k)$ steps (again, for sufficiently large k). □

4.2 Intersection with the Regular Languages

In light of Proposition 10, we now consider the intersection $\mathsf{ACA}(t) \cap \mathsf{REG}$ for $t \in o(n)$ (in the same spirit as a conjecture by Straubing [22]). For this section,

[3] Just as is the case for Turing machines, there is not a single definition for time-constructibility by CAs (see, e.g., [12] for an alternative). Here, we opt for a plausible variant which has the benefit of simplifying the ensuing line of argument.

we assume the reader is familiar with the theory of syntactic semigroups (see, e.g., [7] for an in-depth treatment).

Given a language L, let $\mathrm{SS}(L)$ denote the syntactic semigroup of L. It is well-known that $\mathrm{SS}(L)$ is finite if and only if L is regular. A semigroup S is a *semilattice* if $x^2 = x$ and $xy = yx$ for every $x, y \in S$. Additionally, S is *locally semilattice* if eSe is a semilattice for every *idempotent* $e \in S$, that is, $e^2 = e$. We use the following characterization of locally testable languages:

Theorem 14 ([3,15]). *$L \in \mathsf{LT}$ if and only if $\mathrm{SS}(L)$ is finite and locally semilattice.*

In conjunction with Lemma 9, this yields the following, where the strict inclusion is due to $\mathsf{SOMEONE} \notin \mathsf{ACA}(t)$ (since $\mathsf{SOMEONE} \in \mathsf{LT}$; see Sect. 3):

Theorem 15. *For every $t \in o(n)$, $\mathsf{ACA}(t) \cap \mathsf{REG} \subsetneq \mathsf{LT}$.*

Proof. Let $L \in \mathsf{ACA}(t)$ be a language over the alphabet Σ and, in addition, let $L \in \mathsf{REG}$, that is, $S = \mathrm{SS}(L)$ is finite. By Theorem 14, it suffices to show S is locally semilattice. To that end, let $e \in S$ be idempotent, and let $x, y \in S$.

To show $(exe)(eye) = (eye)(exe)$, let $a, b \in \Sigma^*$ and consider the words $u = a(exe)(eye)b$ and $v = a(eye)(exe)b$. For $m \in \mathbb{N}_+$, let $u'_m = a(e^m x e^m)(e^m y e^m)b$, and let $r \in \mathbb{N}_+$ be such that $r > \max\{|x|, |y|, |a|, |b|\}$ and also $t(|u'_{2r+1}|) < \frac{1}{16|e|}|u'_{2r+1}| < r$. Since e is idempotent, $u' = u'_{2r+1}$ and u belong to the same class in S, that is, $u' \in L$ if and only if $u \in L$; the same is true for $v' = a(e^{2r+1}ye^{2r+1})(e^{2r+1}xe^{2r+1})b$ and v. Furthermore, $p_{2r}(u') = p_{2r}(v')$, $I_{2r+1}(u') = I_{2r+1}(v')$, and $s_{2r}(u') = s_{2r}(v')$ hold. Since $L \in \mathsf{ACA}(t)$, Lemma 11 applies.

The proof of $(exe)(exe) = exe$ is analogous. Simply consider the words $a(e^m x e^m)b$ and $a(e^m x e^m)(e^m x e^m)b$ for sufficiently large $m \in \mathbb{N}_+$ and use, again, Lemma 11 and the fact that e is idempotent. □

Using Theorems 8 and 15, we have $\mathsf{ACA}(t) \subsetneq \mathsf{LT}$ for $t \in o(\log n)$. We can improve this bound to $\mathsf{ACA}(O(1)) = \mathsf{SLT}_\vee$, which is a proper subset of LT:

Theorem 16. *For every $t \in o(\log n)$, $\mathsf{ACA}(t) = \mathsf{ACA}(O(1))$.*

Proof. We prove every ACA C with time complexity at most $t \in o(\log n)$ actually has $O(1)$ time complexity. Let Q be the state set of C and assume $|Q| \geq 2$, and let $n_0 \in \mathbb{N}_+$ be such that $t(n) < \frac{\log n}{9 \log |Q|}$ for $n \geq n_0$. Letting $k(n) = 2t(n)+1$ and assuming $t(n) \geq 1$, we then have $|Q|^{3k(n)} \leq |Q|^{9t(n)} < n$ $(\star)$. We shall use this to prove that, for any word $w \in L$ of length $|w| \geq n_0$, there is a word $w' \in L$ of length $|w'| \leq n_0$ as well as $r < n_0$ such that $p_r(w) = p_r(w')$, $I_{r+1}(w) = I_{r+1}(w')$, and $s_r(w) = s_r(w')$. By Lemma 9, C must have $t(|w'|)$ time complexity on w and, since the set of all such w' is finite, it follows that C has $O(1)$ time complexity.

Now let w be as above and let C accept w in (exactly) $\tau = \tau(w) \leq t(|w|)$ steps. We prove the claim by induction on $|w|$. The base case $|w| = n_0$ is trivial, so let $n > n_0$ and assume the claim holds for every word in L of length strictly less than n. Consider the De Bruijn graph G over the words in $|Q|^\kappa$ where $\kappa = 2\tau+1$. Then, from the infixes of w of length κ (in order of appearance in w) one obtains

a path P in G by starting at the leftmost infix and visiting every subsequent one, up to the rightmost one. Let G' be the induced subgraph of G containing exactly the nodes visited by P, and notice P visits every node in G' at least once. It is not hard to show that, for every such P and G', there is a path P' in G' with the same starting and ending points as P and that visits every node of G' at least once while having length at most $m^2 \leq |Q|^{2\kappa}$, where m is the number of nodes in G'.[4] To this P' corresponds a word w' of length $|w'| \leq \kappa + |Q|^{2\kappa} < |Q|^{3\kappa}$ for which, by construction of P' and G', $p_{\kappa-1}(w') = p_{\kappa-1}(w)$, $I_\kappa(w') = I_\kappa(w)$, and $s_{\kappa-1}(w') = s_{\kappa-1}(w')$. Since $\kappa \leq k(|w|)$, using $(\star)$ we have $|w'| < |w|$, and then either $|w'| \leq n_0$ and $\kappa < n_0$ (since otherwise $w = w'$, which contradicts $|w'| < |w|$), or we may apply the induction hypothesis; in either case, the claim follows. □

4.3 Relation to Parallel Complexity Classes

In this section, we relate $\mathsf{ACA}(t)$ to other classes which characterize parallel computation, namely the SC and (uniform) AC hierarchies. In this context, SC^k is the class of problems decidable by Turing machines in $O((\log n)^k)$ space and polynomial time, whereas AC^k is that decidable by Boolean circuits with polynomial size, $O((\log n)^k)$ depth, and gates with unbounded fan-in. SC (resp., AC) is the union of all SC^k (resp., AC^k) for $k \in \mathbb{N}_0$. Here, we consider only uniform versions of AC; when relevant, we state the respective uniformity condition. Although $\mathsf{SC}^1 = \mathsf{L} \subseteq \mathsf{AC}^1$ is known, it is unclear whether any other containment holds between SC and AC.

One should not expect to include SC or AC in $\mathsf{ACA}(t)$ for any $t \in o(n)$. Conceptually speaking, whereas the models of SC and AC are capable of random access to their input, ACAs are inherently local (as evinced by Lemmas 9 and 11). Explicit counterexamples may be found among the unary languages: For any fixed $m \in \mathbb{N}_+$ and $w_1, w_2 \in \{1\}^+$ with $|w_1|, |w_2| \geq m$, trivially $p_{m-1}(w_1) = p_{m-1}(w_2)$, $I_m(w_1) = I_m(w_2)$, and $s_{m-1}(w_1) = s_{m-1}(w_2)$ hold. Hence, by Lemma 9, if an ACA C accepts $w \in \{1\}^+$ in $t \in o(n)$ time and $|w|$ is large (e.g., $|w| > 4t(|w|)$), then C accepts any $w' \in \{1\}^+$ with $|w'| \geq |w|$. Thus, extending a result from [21]:

Proposition 17. *If $t \in o(n)$ and $L \in \mathsf{ACA}(t)$ is a unary language (i.e., $L \subseteq \Sigma^+$ and $|\Sigma| = 1$), then L is either finite or co-finite.*

In light of the above, the rest of this section is concerned with the converse type of inclusion (i.e., of $\mathsf{ACA}(t)$ in the SC or AC hierarchies). For $f, s, t \colon \mathbb{N}_+ \to \mathbb{N}_0$ with $f(n) \leq s(n)$, we say f is *constructible (by a Turing machine) in $s(n)$ space and $t(n)$ time* if there is a Turing machine T which, on input 1^n, outputs

[4] Number the nodes of G' from 1 to m according to the order in which they are first visited by P. Then, there is a path in G' from i to $i+1$ for every $i \in \{1, \ldots, m-1\}$, and a shortest such path has length at most m. Piecing these paths together along with a last (shortest) path from m to the ending point of P, we obtain a path of length at most m^2 with the purported property.

$f(n)$ in binary using at most $s(n)$ space and $t(n)$ time. Also, recall a Turing machine can simulate τ steps of a CA with m (active) cells in $O(m)$ space and $O(\tau m)$ time.

Proposition 18. *Let C be an ACA with time complexity bounded by $t \in o(n)$, $t(n) \geq \log n$, and let t be constructible in $t(n)$ space and $\mathsf{poly}(n)$ time. Then, there is a Turing machine which decides $L(C)$ in $O(t(n))$ space and $\mathsf{poly}(n)$ time.*

Thus, for polylogarithmic t (where the strict inclusion is due to Proposition 17):

Corollary 19. *For $k \in \mathbb{N}_+$, $\mathsf{ACA}(O((\log n)^k)) \subsetneq \mathsf{SC}^k$.*

Moving on to the AC classes, we employ some notions from descriptive complexity theory (see, e.g., [11] for an introduction). Let $\mathsf{FO_L}[t]$ be the class of languages describable by first-order formulas with numeric relations in L (i.e., logarithmic space) and quantifier block iterations bounded by $t\colon \mathbb{N}_+ \to \mathbb{N}_0$.

Theorem 20. *Let $t\colon \mathbb{N}_+ \to \mathbb{N}_0$ with $t(n) \geq \log n$ be constructible in logarithmic space (and arbitrary time). For any ACA C whose time complexity is bounded by t, $L(C) \in \mathsf{FO_L}[O(\frac{t}{\log n})]$.*

Since $\mathsf{FO_L}[O((\log n)^k)]$ equals L-uniform AC^k [11], by Proposition 17 we have:

Corollary 21. *For $k \in \mathbb{N}_+$, $\mathsf{ACA}(O((\log n)^k)) \subsetneq \mathsf{L}$-uniform AC^{k-1}.*

Because $\mathsf{SC}^1 \not\subseteq \mathsf{AC}^0$ (regardless of non-uniformity) [9], this is an improvement on Corollary 19 at least for $k = 1$. Nevertheless, note the usual uniformity condition for AC^0 is not L- but the more restrictive $\mathsf{DLOGTIME}$-uniformity [25], and there is good evidence that these two versions of AC^0 are distinct [4]. Using methods from [2], Corollary 21 may be rephrased for AC^0 in terms of $\mathsf{TIME}(\mathsf{polylog}(n))$- or even $\mathsf{TIME}((\log n)^2)$-uniformity, but the $\mathsf{DLOGTIME}$-uniformity case remains unclear.

5 Decider ACA

So far, we have considered ACAs strictly as language acceptors. As such, their time complexity for inputs not in the target language (i.e., those which are not accepted) is entirely disregarded. In this section, we investigate ACAs as *deciders*, that is, as machines which must also (explicitly) reject invalid inputs. We analyze the case in which these decider ACAs must reject under the same condition as acceptance (i.e., all cells are simultaneously in a final rejecting state):

Definition 22 (DACA). *A* decider ACA *(*DACA*) is an ACA C which, in addition to its set A of accept states, has a non-empty subset $R \subseteq Q \setminus \{q\}$ of* reject states *that is disjoint from A (i.e., $A \cap R = \varnothing$). Every input $w \in \Sigma^+$ of C must lead to an A- or an R-final configuration (or both). C* accepts *w if it leads*

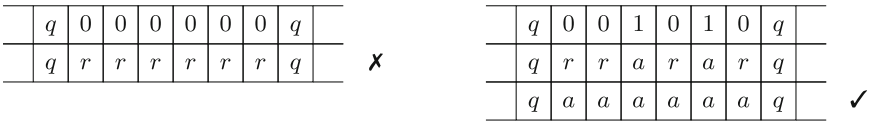

Fig. 2. Computation of a DACA C which decides SOMEONE. The inputs words are $000000 \in L(C)$ and $001010 \notin L(C)$, respectively.

to an A-final configuration c_A and none of the configurations prior to c_A *are R-final. Similarly,* C rejects w *if it leads to an R-final configuration* c_R *and none of the configurations prior to* c_R *are A-final. The time complexity of* C *(with respect to* w*) is the number of steps elapsed until* C *reaches an R- or A-final configuration (for the first time).* $\mathsf{DACA}(t)$ *is the DACA analogue of* $\mathsf{ACA}(t)$.

In contrast to Definition 5, here we must be careful so that the accept and reject results do not overlap (i.e., a word cannot be both accepted and rejected). We opt for interpreting the first (chronologically speaking) of the final configurations as the machine's response. Since the outcome of the computation is then irrelevant regardless of any subsequent configurations (whether they are final or not), this is equivalent to requiring, for instance, that the DACA must halt once a final configuration is reached.

One peculiar consequence of Definition 22 is the relation between languages which can be recognized by acceptor ACAs and DACAs (i.e., the classes $\mathsf{ACA}(t)$ and $\mathsf{DACA}(t)$). As it turns out, the situation is quite different from what is usually expected of restricting an acceptor model to a decider one, that is, that deciders yield a (possibly strictly) more restricted class of machines. In fact, one can show $\mathsf{DACA}(t) \not\subseteq \mathsf{ACA}(t)$ holds for $t \in o(n)$ since $\mathsf{SOMEONE} \notin \mathsf{ACA}(O(1))$ (see discussion after Lemma 9); nevertheless, $\mathsf{SOMEONE} \in \mathsf{DACA}(O(1))$. For example, the local transition function δ of the DACA can be chosen as $\delta(z_1, 0, z_2) = r$ and $\delta(z_1, z, z_2) = a$ for $z \in \{1, a, r\}$, where z_1 and z_2 are arbitrary states, and a and r are the (only) accept and reject states, respectively; see Fig. 2. Choosing the same δ for an (acceptor) ACA does *not* yield an ACA for SOMEONE since then all words of the form 0^+ are accepted in the second step (as they are not rejected in the first one). We stress this rather counterintuitive phenomenon occurs only in the case of sublinear time (as $\mathsf{ACA}(t) = \mathsf{CA}(t) = \mathsf{DACA}(t)$ for $t \in \Omega(n)$).

Similar to (acceptor) ACAs (Lemma 9), sublinear-time DACAs operate locally:

Lemma 23. *Let* C *be a DACA and let* $w \in \{0,1\}^+$ *be a word which* C *decides in exactly* $\tau = \tau(w)$ *steps. Then, for every word* $w' \in \{0,1\}^+$ *with* $p_{2\tau}(w) = p_{2\tau}(w')$, $I_{2\tau+1}(w') = I_{2\tau+1}(w)$, *and* $s_{2\tau}(w) = s_{2\tau}(w')$, C *decides* w' *in* $\leq \tau$ *steps, and* $w \in L(C)$ *holds if and only if* $w' \in L(C)$.

One might be tempted to relax the requirements above to $I_{2\tau+1}(w') \subseteq I_{2\tau+1}(w)$ (as in Lemma 9). We stress, however, the equality $I_{2\tau+1}(w) = I_{2\tau+1}(w')$ is crucial; otherwise, it might be the case that C takes strictly less than τ steps to decide w' and, hence, $w \in L(C)$ may not be equivalent to $w' \in L(C)$.

We note that, in addition to Lemmas 9 and 11, the results from Sect. 4 are extendable to decider ACAs; a more systematic treatment is left as a topic for future work. The remainder of this section is concerned with characterizing $\mathsf{DACA}(O(1))$ computation (as a parallel to Theorem 6) as well as establishing the time threshold for DACAs to decide languages other than those in $\mathsf{DACA}(O(1))$ (as Theorem 16 and the result $\mathsf{BIN} \in \mathsf{ACA}(O(\log n))$ do for acceptor ACAs).

5.1 The Constant-Time Case

First notice that, for any DACA C, swapping the accept and reject states yields a DACA with the same time complexity and which decides the complement of $L(C)$. Hence, in contrast to ACAs (see discussion following Lemma 9):

Proposition 24. *For any $t\colon \mathbb{N}_+ \to \mathbb{N}_+$, $\mathsf{DACA}(t)$ is closed under complement.*

Using this, we can prove the following, which characterizes constant-time DACA computation as a parallel to Theorem 6:

Theorem 25. $\mathsf{DACA}(O(1)) = \mathsf{LT}$.

Hence, we obtain the rather surprising inclusion $\mathsf{ACA}(O(1)) \subsetneq \mathsf{DACA}(O(1))$, that is, for constant time, DACAs constitute a *strictly more* powerful model than their acceptor counterparts.

5.2 Beyond Constant Time

Theorem 16 establishes a logarithmic time threshold for (acceptor) ACAs to recognize languages not in $\mathsf{ACA}(O(1))$. We now turn to obtaining a similar result for DACAs. As it turns out, in this case the bound is considerably larger:

Theorem 26. *For any $t \in o(\sqrt{n})$, $\mathsf{DACA}(t) = \mathsf{DACA}(O(1))$.*

One immediate implication is that $\mathsf{DACA}(t)$ and $\mathsf{ACA}(t)$ are *incomparable* for $t \in o(\sqrt{n}) \cap \omega(1)$ (since, e.g., $\mathsf{BIN} \in \mathsf{ACA}(\log n)$; see Sect. 3). The proof idea is that any DACA whose time complexity is not constant admits an infinite sequence of words with increasing time complexity; however, the time complexity of each such word can be traced back to a critical set of cells which prevent the automaton from either accepting or rejecting. By contracting the words while keeping the extended neighborhoods of these cells intact, we obtain a new infinite sequence of words which the DACA necessarily takes $\Omega(\sqrt{n})$ time to decide:

Proof. Let C be a DACA with time complexity bounded by t and assume $t \notin O(1)$; we show $t \in \Omega(\sqrt{n})$. Since $t \notin O(1)$, for every $i \in \mathbb{N}_0$ there is a w_i such that C takes strictly more than i steps to decide w_i. In particular, when C receives w_i as input, there are cells x_j^i and y_j^i for $j \in \{0, \dots, i\}$ such that x_j^i (resp., y_j^i) is not accepting (resp., rejecting) in step j. Let J_i be the set of all $z \in \{0, \dots, |w_i| - 1\}$ for which $\min\{|z - x_j^i|, |z - y_j^i|\} \leq j$, that is, $z \in N_j(x_j^i) \cup N_j(y_j^i)$ for some j. Consider the restriction w_i' of w_i to the symbols having index in J_i, that is,

$w'_i(k) = w_i(j_k)$ for $J_i = \{j_0, \ldots, j_{m-1}\}$ and $j_0 < \cdots < j_{m-1}$, and notice w'_i has the same property as w_i (i.e., C takes strictly more than i steps to decide w_i). Since $|w'_i| = |J_i| \leq 2(i+1)^2$, C has $\Omega(\sqrt{n})$ time complexity on the (infinite) set $\{w'_i \mid i \in \mathbb{N}_0\}$. □

Using IDMAT (see Sect. 3), we show the bound in Theorem 26 is optimal:

Proposition 27. $\mathsf{IDMAT} \in \mathsf{DACA}(O(\sqrt{n}))$.

We have $\mathsf{IDMAT} \in \mathsf{ACA}(O(\sqrt{n}))$ (see Sect. 3); the non-trivial part is ensuring the DACA also rejects every $w \notin \mathsf{IDMAT}$ in $O(\sqrt{|w|})$ time. In particular, in such strings the # delimiters may be an arbitrary number of cells apart or even absent altogether; hence, naively comparing every pair of blocks is not an option. Rather, we check the existence of a particular set of substrings of increasing length and which must present if the input is in IDMAT. Every $O(1)$ steps the existence of a different substring is verified; the result is that the input length must be at least quadratic in the length of the last substring tested (and the input is timely rejected if it does not contain any one of the required substrings).

6 Conclusion and Open Problems

Following the definition of ACAs in Sect. 2, Sect. 3 reviewed existing results on $\mathsf{ACA}(t)$ for sublinear t (i.e., $t \in o(n)$); we also observed that sublinear-time ACAs operate in an inherently local manner (Lemmas 9 and 11). In Sect. 4, we proved a time hierarchy theorem (Theorem 13), narrowed down the languages in $\mathsf{ACA}(t) \cap \mathsf{REG}$ (Theorem 15), improved Theorem 8 to $\mathsf{ACA}(o(\log n)) = \mathsf{ACA}(O(1))$ (Theorem 16), and, finally, obtained (strict) inclusions in the parallel computation classes SC and AC (Corollaries 19 and 21, respectively). The existence of a hierarchy theorem for ACAs is of interest because obtaining an equivalent result for NC and AC is an open problem in computational complexity theory. Also of note is that the proof of Theorem 13 does not rely on diagonalization (the prevalent technique for most computational models) but, rather, on a quintessential property of sublinear-time ACA computation (i.e., locality as in the sense of Lemma 9).

In Sect. 5, we considered a plausible definition of ACAs as language deciders as opposed to simply acceptors, obtaining DACAs. The respective constant-time class is LT (Theorem 25), which surprisingly is a (strict) superset of $\mathsf{ACA}(O(1)) = \mathsf{SLT}_\vee$. Meanwhile, $\Omega(\sqrt{n})$ is the time complexity threshold for deciding languages other than those in LT (Theorem 26 and Proposition 27).

As for future work, the primary concern is extending the results of Sect. 4 to DACAs. $\mathsf{DACA}(O(1)) = \mathsf{LT}$ is closed under union and intersection and we saw that $\mathsf{DACA}(t)$ is closed under complement for any $t \in o(n)$; a further question would be whether $\mathsf{DACA}(t)$ is also closed under union and intersection. Finally, we have $\mathsf{ACA}(O(1)) \subsetneq \mathsf{DACA}(O(1))$, $\mathsf{ACA}(O(n)) = \mathsf{CA}(O(n)) = \mathsf{DACA}(O(n))$, and that $\mathsf{ACA}(t)$ and $\mathsf{DACA}(t)$ are incomparable for $t \in o(\sqrt{n}) \cap \omega(1)$; it remains open what the relation between the two classes is for $t \in \Omega(\sqrt{n}) \cap o(n)$.

Acknowledgments. I would like to thank Thomas Worsch for the fruitful discussions and feedback during the development of this work. I would also like to thank the DLT 2020 reviewers for their valuable comments and suggestions and, in particular, one of the reviewers for pointing out a proof idea for Theorem 16, which was listed as an open problem in a preliminary version of the paper.

References

1. Arora, S., Barak, B.: Computational Complexity - A Modern Approach. Cambridge University Press, Cambridge (2009)
2. Mix Barrington, D.A.: Extensions of an idea of McNaughton. Math. Syst. Theory **23**(3), 147–164 (1990). https://doi.org/10.1007/BF02090772
3. Brzozowski, J.A., Simon, I.: Characterizations of locally testable events. Discrete Math. **4**(3), 243–271 (1973). https://doi.org/10.1016/S0012-365X(73)80005-6
4. Caussinus, H., et al.: Nondeterministic NC^1 computation. J. Comput. Syst. Sci. **57**(2), 200–212 (1998). https://doi.org/10.1006/jcss.1998.1588
5. Cook, S.A.: A taxonomy of problems with fast parallel algorithms. Inf. Control **64**(1–3), 2–21 (1985). https://doi.org/10.1016/S0019-9958(85)80041-3
6. Delorme, M., Mazoyer, J. (eds.): Cellular Automata. A Parallel Model. Mathematics and Its Application, vol. 460. Springer, Netherlands (1999). https://doi.org/10.1007/978-94-015-9153-9
7. Eilenberg, S.: Automata, Languages, and Machines. Pure and Applied Mathematics, vol. B. Academic Press, New York (1976)
8. Fischer, E.: The art of uninformed decisions. In: Bulletin of the EATCS 75, p. 97 (2001)
9. Furst, M.L., et al.: Parity, circuits, and the polynomial-time hierarchy. Math. Syst. Theory **17**(1), 13–27 (1984). https://doi.org/10.1007/BF01744431
10. Ibarra, O.H., et al.: Fast parallel language recognition by cellular automata. Theor. Comput. Sci. **41**, 231–246 (1985). https://doi.org/10.1016/0304-3975(85)90073-8
11. Immerman, N.: Descriptive Complexity. Texts in Computer Science. Springer, New York (1999). https://doi.org/10.1007/978-1-4612-0539-5
12. Iwamoto, C., et al.: Constructible functions in cellular automata and their applications to hierarchy results. Theor. Comput. Sci. **270**(1–2), 797–809 (2002). https://doi.org/10.1016/S0304-3975(01)00112-8
13. Kim, S., McCloskey, R., Sam Kim and Robert McCloskey: A characterization of constant-time cellular automata computation. Phys. D **45**(1–3), 404–419 (1990). https://doi.org/10.1016/0167-2789(90)90198-X
14. Kutrib, M.: Cellular automata and language theory. In: Meyers, R. (ed.) Encyclopedia of Complexity and Systems Science, pp. 800–823. Springer, New York (2009). https://doi.org/10.1007/978-0-387-30440-3
15. McNaughton, R.: Algebraic decision procedures for local testability. Math. Syst. Theory **8**(1), 60–76 (1974). https://doi.org/10.1007/BF01761708
16. McNaughton, R., Papert, S.: Counter-Free Automata. The MIT Press, Cambridge, MA (1971)
17. Modanese, A.: Sublinear-Time Language Recognition and Decision by One-Dimensional Cellular Automata. CoRR abs/1909.05828 (2019). arXiv: 1909.05828
18. Rosenfeld, A.: Picture Languages: Formal Models for Picture Recognition. Academic Press, New York (1979)

19. Rubinfeld, R., Shapira, A., Ronitt Rubinfeld and Asaf Shapira: Sublinear time algorithms. SIAM J. Discrete Math. **25**(4), 1562–1588 (2011). https://doi.org/10.1137/100791075
20. Ruzzo, W.L.: On uniform circuit complexity. J. Comput. Syst. Sci. **22**(3), 365–383 (1981). https://doi.org/10.1016/0022-0000(81)90038-6
21. Sommerhalder, R., van Westrhenen, S.C.: Parallel language recognition in constant time by cellular automata. Acta Inf. **19**, 397–407 (1983). https://doi.org/10.1007/BF00290736
22. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Progress in Theoretical Computer Science. Birkhäuser, Boston, MA (1994). https://doi.org/10.1007/978-1-4612-0289-9
23. Sudan, M.: Probabilistically checkable proofs. Commun. ACM **52**(3), 76–84 (2009). https://doi.org/10.1145/1467247.1467267
24. Terrier, V.: Language recognition by cellular automata. In: Rozenberg, G., Back, T., Kok, J.N. (eds.) Handbook of Natural Computing, pp. 123–158. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-540-92910-9_4
25. Vollmer, H.: Introduction to Circuit Complexity - A Uniform Approach. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-662-03927-4