From $k+1$ to k heads the descriptive trade-off is non-recursive

Christos Kapoutsis

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
cak@mit.edu

Abstract

We prove that no recursive function can upper bound the increase in the size of description when a two-way deterministic finite automaton with $k+1$ heads is replaced by an equivalent two-way deterministic finite automaton with k heads. This is true for all k, and remains true if the automata are unary and/or nondeterministic.

1 Introduction

Let 2DFA_k stand for 'two-way deterministic finite automaton with k heads' and (2DFA_k) for the set of languages recognizable by such automata. It is well known that the sequence

$$(\text{2DFA}_1),\ (\text{2DFA}_2),\ \ldots,\ (\text{2DFA}_k),\ \ldots$$

starts with the regular languages [13, 16], is strictly increasing [10, 11], and eventually covers the entire class $\mathbf{L}$ of languages decidable by Turing machines (TMs) in logarithmic space [2]. Hence, for any $L \in \mathbf{L}$ and for k^* the smallest k for which $L \in (\text{2DFA}_k)$, each one of the following types of machines

$$\text{2DFA}_{k^*},\ \text{2DFA}_{k^*+1},\ \text{2DFA}_{k^*+2},\ \ldots\quad,\ \text{logarithmic-space TM},$$

suggests a different string to 'describe' L: a smallest (description of a) machine of this type that recognizes L. It is then conceivable that stronger types of machines may provide more succinct descriptions than weaker types, making it natural to ask how much more verbose the weaker types can be, in general.

Recently, Kutrib [5] proved that 2DFA_ks can be both non-recursively more verbose than logarithmic-space TMs and non-recursively more succinct than 2DFA_1s. That is, he showed that no recursive function can upper bound the increase in the size of description when we replace a minimal (description of a) logarithmic-space Turing machine with (a description of) an equivalent 2DFA_k, for any $k \geq 1$. Similarly, on the other end of the hierarchy, the increase in the size of description when a minimal (description of a) 2DFA_k is replaced by (a description of) an equivalent 2DFA_1 admits no recursive upper bound, either, for any $k \geq 2$.

Kutrib went on to conjecture that the trade-off is non-recursive even when we change from a minimal 2DFA_{k+1} into an equivalent 2DFA_k, for all k. In addition, he posed the question whether the same is true when we restrict our attention to unary automata of this type —for this case he found huge recursive lower bounds but refrained from guessing that no recursive upper bound exists, because of the real-time one-way cellular automata precedent (for these automata, non-recursive trade-offs for arbitrary languages reduce to recursive trade-offs for unary languages [6]).

We resolve these questions by showing that non-recursive blow-ups are possible both in the general and in the unary case. Moreover, if we switch our attention to nondeterministic automata, the blow-ups can still be non-recursive, again both in the general and in the unary case. Section 2 describes the framework of this study in more detail and gives an outline of the proof, whose two pieces are presented in Sections 3 and 5. The discussion focuses on the general deterministic case, which is the most natural one; when appropriate, a final note in each section addresses the unary and/or nondeterministic cases.

Related work. The study of the relative succinctness of descriptional systems of different power seems to have started with Stearns [18], who gave a recursive upper bound for the size of the smallest deterministic finite automaton (1DFA) that recognizes the language of a given deterministic pushdown automaton (1DPA) whenever this language is indeed regular. Following this, Meyer and Fischer [7] showed no such bound exists if we compare 1DFAs to context-free grammars —the first proof of a non-recursive trade-off. Later, Valiant [19] proved the same for the comparison between 1DPAs and unambiguous context-free grammars, while Schmidt and Szymanski [15] repeated this for the comparison between the latter and (general) context-free grammars. Finally, Hartmanis [3] simplified many of the proofs by noting the relation of the recursiveness of a trade-off to the recognizability of the corresponding inadequacy problem (cf. Lemma 2.3); he also proved non-recursive trade-offs even for comparisons where the nondeterminstic descriptors from above are accompanied by proofs that their languages are deterministic or unambiguous. It is this discussion that Kutrib [5] has transferred to multihead finite automata.

For a detailed overview of the subject, along with a comprehensive survey of the broader field of descriptive complexity, the reader is referred to [1].

2 The framework

We denote the set of positive integers by $\mathbb{N}$. For $n, a \in \mathbb{N}$, we write $\lg_a n$ to denote $\lfloor \log_a n \rfloor$. The set of all finite strings over an alphabet Γ is denoted by Γ^*.

A (*promise*) *problem* over Γ is any pair $\Pi = (\Pi_{\text{yes}}, \Pi_{\text{no}})$ where Π_{yes}, Π_{no} are disjoint subsets of Γ^*. A TM *recognizes* Π if it accepts all $x \in \Pi_{\text{yes}}$ and rejects all $x \in \Pi_{\text{no}}$ —possibly by looping. If some TM recognizes Π, we say Π is *recursively enumerable* or (*Turing-*) *recognizable*. For Π' also a problem over Γ, we write $\Pi \leq \Pi'$ and say Π *reduces to* Π' iff there is a TM that, on input $x \in \Gamma^*$, eventually halts with an output y such that $x \in \Pi_{\text{yes}} \implies y \in \Pi'_{\text{yes}}$ and $x \in \Pi_{\text{no}} \implies y \in \Pi'_{\text{no}}$. If some unrecognizable Π reduces to Π', then Π' is also unrecognizable [4, 17].

If $\Pi_{\text{no}} = \overline{\Pi_{\text{yes}}}$, then Π is also called a *language* and is adequately described by Π_{yes} alone. If in addition Π_{yes} contains exactly all sufficiently long strings for some interpretation l of 'sufficiently long', $\Pi_{\text{yes}} = \{x \in \Gamma^* \mid \text{length}(x) \geq l\}$, then we say Π is a *threshold language*.

2.1 Multihead automata

A *two-way deterministic finite automaton with k heads* (a 2DFA_k) *over alphabet* $\Sigma \neq \emptyset$ consists of a finite state control, k read-only heads, and a tape that can represent the symbols of Σ and the extra endmarking symbols $\vdash$ and $\dashv$. An input $x \in \Sigma^*$ is presented on the tape surrounded by the endmarkers. The automaton starts at a designated *start state*, all heads reading the left endmarker $\vdash$. At every step, the k symbols under the heads are read; based on this information and the current state, the automaton decides what its next state should be, which one head it should move, and whether to move it left or right; it then simultaneously changes its state and moves the chosen head accordingly. No head can move past an endmarker and the automaton cannot sense when two of its heads are on the same tape cell. The input is *accepted* if the machine ever reaches a designated *final state* and halts. The *language* of the automaton is exactly the set of strings over Σ that it accepts. We say the automaton is *terminating* if it halts on all inputs.

By a *multihead two-way deterministic finite automaton* (a 2DFA_*), we mean a 2DFA_k with any number of heads k. A *description* of such an automaton is any string that encodes it, under some arbitrary (but fixed) reasonable encoding of 2DFA_*s into finite strings over some alphabet Δ. The *size* of such a description $z \in \Delta^*$ is the value $\text{size}(z)$, where $\text{size}\colon \Delta^* \to \mathbb{N}$ is some arbitrary (but fixed) computable function with the property that some terminating TM can, on input $k, s \in \mathbb{N}$, produce a list of descriptions that cover all 2DFA_ks with a description of size at most s.

We write (z) for the 2DFA_* encoded by a description z, and (M) for the language of a 2DFA_* M; the set of all descriptions of M (which may not be a singleton) is denoted by $\langle M \rangle$. The collection of the languages of all 2DFA_ks [all 2DFA_*s] is denoted by (2DFA_k) [by (2DFA_*)]. The collection of the descriptions of all 2DFA_ks [all 2DFA_*s] is denoted by $\langle \text{2DFA}_k \rangle$ [by $\langle \text{2DFA}_* \rangle$]. Two 2DFA_*s are *equivalent* if their languages are identical; two descriptions $z, z' \in \langle \text{2DFA}_* \rangle$ are equivalent if (z), (z') are.

The unary and nondeterministic cases. If Σ in the definition of a 2DFA_k contains only one symbol, we say the automaton is *unary* (a 2DFA_k^{u}). If more than one next moves are allowed at each step and the input is considered to be accepted whenever at least one of the resulting computations halts at a final state, we say the automaton is *nondeterministic* (a 2NFA_k; or a 2NFA_k^{u}, if the alphabet is unary). The sets of languages $(\text{2DFA}_k^{\text{u}})$, (2NFA_*), etc. and descriptions $\langle \text{2DFA}_k^{\text{u}} \rangle$, $\langle \text{2NFA}_* \rangle$, etc. are defined analogously.

2.2 Decreasing the number of heads

It is known [10, 11] that 2DFA_{k+1}s are more powerful than 2DFA_ks, in the sense that

$$(\text{2DFA}_k) \subsetneqq (\text{2DFA}_{k+1}), \tag{1}$$

for all k. This suggests two natural discussions.

I. Since $(\text{2DFA}_k) \subseteq (\text{2DFA}_{k+1})$, an $L \in (\text{2DFA}_k)$ can be 'described' not only by descriptions of 2DFA$_k$s that recognize it, but also by descriptions of 2DFA$_{k+1}$s. Conceivably, the minimal descriptions of the former type are larger than the minimal descriptions of the latter type, in which case to change from a minimal 2DFA$_{k+1}$ description of L to a 2DFA$_k$ description of L is to trade an increase in the size of the description for a decrease in the number of heads. It is then natural to ask for an upper bound for this increase.

2.1 Definition. If $f_k : \mathbb{N} \to \mathbb{N}$ is such that

> for all $s \in \mathbb{N}$ and all $z \in \langle \text{2DFA}_{k+1} \rangle$ of size at most s: if z has an equivalent $z' \in \langle \text{2DFA}_k \rangle$, then it has one of size at most $f_k(s)$,

then we say that f_k is an upper bound for the increase in the size of description when a 2DFA$_{k+1}$ is replaced by an equivalent 2DFA$_k$.

So, if f_k is such an upper bound and M is a 2DFA$_{k+1}$ that has equivalent 2DFA$_k$s, at least one of these has a small enough description: no larger than $f_k(s)$, where s is the size of any description of M. To define such an f_k is, of course, very easy; what is questionable is whether we can define one that is also recursive. If this is not possible, we say that *the* (*descriptive*) *trade-off from* $k+1$ *to* k *heads is non-recursive* (*for* 2DFA$_*$s), or that 2DFA$_{k+1}$s *can be non-recursively more succinct than* 2DFA$_k$s.

II. Since $(\text{2DFA}_k) \neq (\text{2DFA}_{k+1})$, a 2DFA$_{k+1}$ may or may not have an equivalent 2DFA$_k$, so that it is natural to consider the associated computational problem: "Given a description of a 2DFA$_{k+1}$ M, is it true that M has no equivalent 2DFA$_k$?"

2.2 Definition. $\mathsf{H}_k = \big\{ z \in \langle \text{2DFA}_{k+1} \rangle \mid \neg\big(\exists z' \in \langle \text{2DFA}_k \rangle\big)\big(z,\, z' \text{ are equivalent}\big) \big\}$.

In a more intuitive dialect, H_k is the problem: "Given a 2DFA$_{k+1}$ M, is it true that M can't perform the same task with only k of its heads?" We are interested in the question whether this problem is recognizable, in which case we say that *the inadequacy of* k *heads is recognizable* (*for* 2DFA$_{k+1}$s).

As first noted in [3], discussions (I) and (II) are not unrelated (see also [5]).

2.3 Lemma. *If the trade-off from* $k+1$ *to* k *heads is recursive, then the inadequacy of* k *heads is recognizable.*

The unary and nondeterministic cases. That $k+1$ heads are better than k has also been established for the case where the automata are unary and/or nondeterministic [11, 12]. That is, similarly to (1), we also have

$$(\text{2DFA}^{\text{u}}_k) \subsetneqq (\text{2DFA}^{\text{u}}_{k+1}), \qquad (\text{2NFA}_k) \subsetneqq (\text{2NFA}_{k+1}), \qquad (\text{2NFA}^{\text{u}}_k) \subsetneqq (\text{2NFA}^{\text{u}}_{k+1}),$$

for all k. As a result, for each one of the three types of machines we can repeat the preceding discussions, asking again (I) whether the trade-off from $k+1$ to k heads is recursive, and (II) whether the inadequacy of k heads is recognizable. Lemma 2.3 can then be shown anew (with the same proof).

2.3 Proof outline

Fix any $k \geq 1$. Our goal is to prove that 2DFA_{k+1}s can be non-recursively more succinct than 2DFA_ks. By Lemma 2.3, it is enough to show that H_k is unrecognizable. We do this by a reduction from the complement of the halting problem

$$\overline{\mathsf{HALTING}} \leq \mathsf{H}_k,$$

where $\overline{\mathsf{HALTING}} = \{z \in \{0,1\}^* \mid z \text{ encodes a TM that loops on } z\}$ is known to be unrecognizable [4, 17]. What we actually show is two separate reductions:

$$\overline{\mathsf{HALTING}} \leq \mathsf{E} \leq \mathsf{H}_k, \tag{2}$$

where E is the promise problem: "Given a description of a terminating 2DFA^{u}_2 whose language is either empty or threshold, check that it is empty." In formal dialect, $\mathsf{E} = (\mathsf{E}_{\text{yes}}, \mathsf{E}_{\text{no}})$, where

$$\begin{aligned}\mathsf{E}_{\text{yes}} &= \{z \in \langle \text{2DFA}^{\text{u}}_2 \rangle \mid (z) \text{ is terminating and } ((z)) = \emptyset\},\\ \mathsf{E}_{\text{no}} &= \{z \in \langle \text{2DFA}^{\text{u}}_2 \rangle \mid (z) \text{ is terminating and } ((z)) \text{ is a threshold language}\}.\end{aligned}$$

The next section discusses the second of the two reductions in (2), which is easier. The first reduction is established in Section 5 and makes use of multicounter automata, as introduced in Section 4.

The unary and nondeterministic cases. Lemma 2.3 again suggests that we need only prove the inadequacy of k heads is unrecognizable, for each one of the three types of automata. We do this with the same two reductions of (2). Only the second one needs to be modified.

3 The second reduction

Our goal in this section is to find an algorithm that, on input a description z of a terminating 2DFA^{u}_2 M whose language is guaranteed to be either empty or threshold, produces a description z' of a 2DFA_{k+1} M' such that

$$\begin{aligned}(M) = \emptyset &\implies M' \text{ has no equivalent } \text{2DFA}_k,\\ (M) \text{ is a threshold language} &\implies M' \text{ has equivalent } \text{2DFA}_k\text{s}.\end{aligned} \tag{3}$$

Following the idea of [5, Theorem 7], we will need an arbitrary 2DFA_{k+1} that has no equivalent 2DFA_k. So, fix D to be such a 2DFA_{k+1} and z_D to be a description of it; also let Σ be D's input alphabet.

Our algorithm is simple. It uses z and z_D to construct a description z' for the 2DFA_{k+1} M' that, on input $x \in \Sigma^*$, follows the instructions:

1. Simulate M on x; if M accepts, accept; otherwise, go to Step 2.
2. Simulate D on x (and accept, reject, or loop accordingly).

Note that in the first simulation M' treats all symbols of Σ the same. Also, this simulation *is* possible, since $k + 1 \geq 2$. Moreover, when M rejects, M' *can* move to Step 2, since M never rejects by looping. Overall, M' is well-defined.

As for the language of M', it is easy to see that: when $(M) = \emptyset$, then (M') is just (D); when (M) is a threshold language with threshold l, then

$$(M') = \{x \in (D) \mid \text{length}(x) < l\} \quad \cup \quad \{x \in \Sigma^* \mid \text{length}(x) \geq l\},$$

which is cofinite, and hence regular. In total,

$$(M) = \emptyset \Rightarrow (M') = (D) \qquad \& \qquad (M) \text{ is threshold} \Rightarrow (M') \text{ is regular}, \qquad (4)$$

which implies (3), by the selection of D and the inclusion of regular languages in $(\text{2DFA}_1) \subseteq (\text{2DFA}_k)$. Hence, our algorithm can safely output z'.

The unary and nondeterministic cases. By appropriately changing the selection of z_D, the same algorithm establishes that $\mathsf{E} \leq \mathsf{H}_k$, even when H_k stands for the inadequacy of k heads for $\text{2DFA}^{\text{u}}_{k+1}$s [$\text{2NFA}_{k+1}$s, $\text{2NFA}^{\text{u}}_{k+1}$s]. More specifically, we now select z_D to be a description of a $\text{2DFA}^{\text{u}}_{k+1}$ [2NFA_{k+1}, $\text{2NFA}^{\text{u}}_{k+1}$] D that has no equivalent $\text{2DFA}^{\text{u}}_{k}$ [2NFA_{k}, $\text{2NFA}^{\text{u}}_{k}$]. Then, M' is a well-defined automaton of the same type as D (note that M' has enough resources to simulate each of M, D) that satisfies (4), and hence the version of (3) for $\text{2DFA}^{\text{u}}_{k}$s [$\text{2NFA}_{k}$s, $\text{2NFA}^{\text{u}}_{k}$s].

4 Multicounter automata

A *deterministic automaton with k (bounded) counters*[1] (DCA_k) consists of a finite state control and k counters, each of which can store a nonnegative integer. The input to the automaton is a nonnegative upper bound n for its counters. The automaton starts at a designated *start state* with all its counters set to 0. At every step, based on its current state, the automaton decides which counter it should act upon and whether it should decrease it or increase it. Then the action is *attempted*. An attempt to decrease fails iff the counter already contains 0; an attempt to increase fails iff the counter already contains n. A failed attempt leaves the counter contents intact; a successful attempt updates the counter contents accordingly. Based on its current state and on whether the attempt succeeded or not, the automaton selects a new state and moves to it. The input is *accepted* if the machine ever enters a designated *final state*. The *language* of the automaton is exactly the set of nonnegative integers that it accepts. By a *deterministic multicounter automaton* (a DCA_*) we mean a DCA_k with any number of counters k.

The new automata are just another, more convenient way to look at $\text{2DFA}^{\text{u}}_{*}$s.

4.1 Lemma. *Every* $\text{2DFA}^{\text{u}}_{k}$ M *has a* DCA_k M' *such that* $(M') = \{n+1 \mid 0^n \in (M)\}$. *Conversely, every* DCA_k M *has a* $\text{2DFA}^{\text{u}}_{k}$ M' *with* $(M') = \{0^{n-1} \mid n \in (M) \ \& \ n \neq 0\}$.

[1] In this definition the reader will probably almost recognize the *bounded counting automata* of [14], the *two-way counter automata* and the *register machines* of [12], or other automata from elsewhere. The present model is nicely simpler, in the sense that the input is directly taken to be the upper bound for the counters, saving us the redundant (in this case) notion of an input tape or an input register.

In order to facilitate the description of the behavior of DCA$_*$s, we introduce some 'program' notation. First, the two atomic operations, the attempt to decrease a counter X and the attempt to increase it, are denoted respectively by

$$X \xleftarrow{f} X-1 \qquad \text{and} \qquad X \xleftarrow{f} X+1,$$

where, in each case, flag f is set to true iff the attempt succeeds. Then, the compound operation of setting X to 0, denoted by $X \longleftarrow 0$, can be described by the line

$$\texttt{repeat}\ \ X \xleftarrow{f} X-1\ \ \texttt{until}\ \neg f. \tag{5}$$

If a second counter Y is present, we can transfer the contents of Y into X: we set X to 0, then repeatedly decrease Y and increase X until Y is 0. We denote this by

$$(X,Y) \longleftarrow (Y,0),$$

and describe it by a line similar to (5). Changing how fast X increases as Y decreases, we can multiply/divide Y into X by any constant a. We denote these operations by

$$(X,Y) \xleftarrow{f} (aY,0) \qquad \text{and} \qquad (X,Y) \xleftarrow{r} \big(\lfloor \tfrac{Y}{a} \rfloor, 0\big).$$

For the first one, note that $aY > n$ implies one of the attempts to increase X will fail; in that case, we restore the original value of Y returning X to 0, and set flag f to false. In the second operation, we also find the remainder and return it in r.

At a higher level, we can attempt to multiply Y by a constant a (into Y) using X as an auxiliary counter and making sure Y changes only if the operation succeeds:

$$(X,Y) \xleftarrow{f} (aY,0);\quad \texttt{if}\ f\ \texttt{then}\ (Y,X) \longleftarrow (X,0).$$

Division (with remainder) can be done similarly. We denote the two operations by

$$Y \xleftarrow{f,X} aY \qquad \text{and} \qquad Y \xleftarrow{r,X} \lfloor \tfrac{Y}{a} \rfloor. \tag{6}$$

Now, we can set Y to the largest power of a that can fit in n:

$$Y \longleftarrow 0;\ Y \xleftarrow{f} Y+1;\ \texttt{if}\ f\ \texttt{then repeat}\ Y \xleftarrow{g,X} aY\ \texttt{until}\ \neg g \tag{7}$$

(note that this fails iff $n = 0$). We denote (7) by (remember that $\lg_a n = \lfloor \log_a n \rfloor$):

$$Y \xleftarrow{f,X} a^{\lg_a n}.$$

If a third counter Z is present, we can modify (7) to also count (in Z) the number of iterations performed. This gives us a way to try to calculate $\lg_a n$ (failing iff $n = 0$):

$$Z \xleftarrow{f,X,Y} \lg_a n.$$

In another variation, we can modify the multiplication in (6) so that the success of the operation depends on the contents of Z (as opposed to n). We write this as

$$Y \xleftarrow{f,X,Z} aY,$$

meaning that, using X as auxiliary and without affecting Z: if $aY \le Z$, then Y is set to aY; otherwise, Y is unaffected.[2] Then, the following variant of (7)

$$Z \xleftarrow{f} Z-1;\ \texttt{if}\ f\ \texttt{then}\ \{Z \xleftarrow{\texttt{t}} Z+1;\ \texttt{repeat}\ Y \xleftarrow{g,X,Z} aY\ \texttt{until}\ \neg g\}$$

implements the attempt to set Y to the largest power of a that is at most Z, using X as auxiliary and leaving Z unaffected (failing iff Z is 0).[3] We denote this one by

$$Y \xleftarrow{f,X} a^{\lg_a Z}.$$

It is important to note that *during this operation no counter ever assumes a value greater than the original value of Z* (cf. Footnote 2).

Hopefully, the reader is convinced of the quite significant capabilities of DCA$_*$s that have 2 or more counters. We will be using these capabilities in the next section.

5 The first reduction

In this section, our goal is to find an algorithm that, on input a decription z of a TM M produces a description z' of a terminating 2DFA$_2^{\text{u}}$ M' such that

$$\begin{aligned} &M \text{ loops on input } z \implies (M') = \emptyset, \\ &M \text{ halts on input } z \implies (M') \text{ is a threshold language.} \end{aligned} \tag{8}$$

We will be calling a machine (TM, DCA$_*$, or 2DFA$_*^{\text{u}}$) *good* if it is terminating and its language satisfies (8) when it replaces (M'). For example, (z') should be good.

On its way to z', our algorithm will construct descriptions of three other machines, in this order: a description z_A of a TM A, a description z_B of a DCA$_3$ B, and a description z_C of a DCA$_2$ C. In the sequence M, A, B, C, M', each machine after M will be defined in terms of the previous one and will be good. The reader will probably recognize in our constructions the ideas of [20] and [8] (also found in [9, 4]).

The first machine. A is a TM with one tape, infinite in both directions; the tape alphabet is $\{\sqcup, 0, 1, \dot{0}, \dot{1}\}$, while the input alphabet is $\{0\}$. On input 0^n, A starts with tape contents

$$\cdots \sqcup\sqcup\sqcup \underbrace{000\cdots00}_{n \text{ times}} \sqcup\sqcup\sqcup\cdots$$

and its head reading the leftmost 0 (or a $\sqcup$, if $n = 0$). It then computes:

1. For all $x \in \{0,1\}^n$, from 0^n up to 1^n:
   — if x encodes a halting computation history of M on z, accept.
2. Reject.

[2] To implement this, we repeatedly decrease Y, increase X, and decrease Z by a. *If Z becomes 0 before Y*, then $aY > Z$ and the operation should fail: we restore the original values of Y and Z by repeatedly decreasing X, increasing Y, and increasing Z by a, until X becomes 0. *Otherwise*, $aY \le Z$ and the operation will succeed: we copy the correct value to Y and restore the value of Z by repeatedly decreasing X and increasing each of Y, Z by a, until X becomes 0. Note that *none of the counters ever assumes a value greater than the original value of Z.*

[3] Note the use of t in the place of a flag, indicating that the action is guaranteed to succeed.

The check inside the loop presupposes some fixed reasonable encoding of sequences of configurations of M into binary strings, with the additional property: *if w encodes a computation history, then every string of the form $w0^*$ encodes the same history.* Note that, using the extra dotted symbols, A can easily perform this check without ever writing a non-blank symbol on a $\sqcup$, or a $\sqcup$ on a non-blank symbol. As a consequence, throughout its computation on 0^n, A keeps exactly n non-blank symbols on its tape (occupying the same n cells as the symbols of the input). This will prove useful in the next section.

The second machine. B is a DCA$_3$ that, on input $n \geq 30$, simulates the behavior of A on input $0^{\lg_5 \lg_{30} n}$; on input $n < 30$, B just rejects.[4]

To explain B's behavior, let L, R, J be its three counters. J is auxiliary, used for performing operations on L and R, while L and R together encode tape configurations of A. To see the encoding, consider the following example of a configuration:

$\cdots$	l_4	l_3	l_2	l_1	l_0	r_0	r_1	r_2	r_3	r_4	r_5	r_6	$\cdots$
$\cdots$	$\sqcup$	$\sqcup$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\sqcup$	$\sqcup$	$\cdots$
						$\uparrow$							

(here $\times$ stands for any non-blank symbol, $\uparrow$ shows the head position). Mapping symbols $\sqcup$, $\dot{1}$, $\dot{0}$, 1 and 0 to numbers 0, 1, 2, 3 and 4, respectively, we get each tape cell map to a digit of the 5-ary numbering system. Then, the head position splits the tape into two portions, which define the integers

$$l = \sum_{i=0}^{\infty} l_i \cdot 5^i \qquad \text{and} \qquad r = \sum_{r=0}^{\infty} r_i \cdot 5^i$$

(note that the cell under the head contributes to r; also, the sums are finite exactly because $\sqcup$ maps to 0). These two values are kept in L and R, respectively.

More specifically, on input n, B starts with a two-part initialization. First, it computes $\lg_{30} n$ into J, leaving 0s in L and R (this is if $n \geq 1$; if $n = 0$, it rejects):

$$L \overset{f,R,J}{\longleftarrow} \lg_{30} n;\ \texttt{if}\ \neg f\ \texttt{then reject else}\ (J, L) \longleftarrow (L, 0),$$

Then, it computes into R the value $5^m - 1$, where $m = \lg_5 \lg_{30} n$, leaving 0 in L (this is only if $J \geq 1$, that is if $n \geq 30$; otherwise, $n < 30$ and B rejects):

$$R \overset{f,L}{\longleftarrow} 5^{\lg_5 J};\ \texttt{if}\ \neg f\ \texttt{then reject else}\ \{R \overset{\texttt{t}}{\longleftarrow} R - 1;\ L \longleftarrow 0\}.$$

This completes the initialization, with $L = 0$ and $R = 5^m - 1$, or in 5-ary:

$$L = 0 \qquad \text{and} \qquad R = \underbrace{4\ 4\ 4\ \cdots\ 4}_{m \text{ times}}.$$

Hence L and R correctly represent the starting tape configuration of A on input 0^m:

[4] The strange length $\lg_5 \lg_{30} n$ of the input of A's simulated computation is chosen as a function of n that is (i) *computable* by a DCA$_3$ and (ii) *increasing*, but also (iii) *small enough*. Goodness of B bases on (ii), while (iii) facilitates the simulation performed by C in the next section.

$\cdots$	l_1	l_0	r_0	r_1	r_2	$\cdots$	r_{m-1}	r_m	r_{m+1}	$\cdots$
$\cdots$	$\sqcup$	$\sqcup$	0	0	0	$\cdots$	0	$\sqcup$	$\sqcup$	$\cdots$
			$\uparrow$							

(since symbol 0 maps to 4) and B is ready to start a faithful simulation of A's steps.

The automaton remembers the current state of A in its finite memory. To find the code r_0 of the symbol read by the head of A, it computes:

$$R \xleftarrow{r_0,J} \left\lfloor \tfrac{R}{5} \right\rfloor.$$

Then, if s is the code of the symbol to be written on the tape, it computes

$$L \xleftarrow{\mathtt{t},J} 5L;\ \texttt{repeat}\ s\ \texttt{times:}\ L \xleftarrow{\mathtt{t}} L+1$$

to simulate writing this symbol and moving to the right; it computes

$$R \xleftarrow{\mathtt{t},J} 5R;\ \texttt{repeat}\ s\ \texttt{times:}\ R \xleftarrow{\mathtt{t}} R+1;$$
$$L \xleftarrow{l_0,J} \left\lfloor \tfrac{L}{5} \right\rfloor;\ R \xleftarrow{\mathtt{t},J} 5R;\ \texttt{repeat}\ l_0\ \texttt{times:}\ R \xleftarrow{\mathtt{t}} R+1$$

to simulate writing this symbol and moving to the left.[5]

It is important to note the range of the values assumed by the counters. By the design of its main operation, the second part of the initialization phase never assigns to a counter a value greater than the original value of J, which is $\lg_{30} n$. Then, in the simulation phase, the behavior of A (the tape starts with m 0's and always contains exactly m non-blank symbols) and the selection of the symbol codes (0 gets the largest code) are such that the initial value $5^m - 1$ of R upper bounds all possible values that may appear in B's counters. One consequence of this is that all increments and multiplications in the previous paragraph are guaranteed to be successful (hence the `t` reminder). Another consequence is that, since $5^m - 1 < \lg_{30} n$, *the entire computation of B after the first part of its initialization phase keeps all values of all counters at or below* $\lg_{30} n$. This will prove crucial in the next section.

The third machine. C is a DCA_2 that simulates the behavior of B. If U, V are its two counters, then V is auxiliary for performing operations on U, while U encodes the contents of the counters of B: whenever L, R, J contain l, r, j respectively, U contains $2^l 3^r 5^j$.

The automaton starts by computing into U the product $30^t = 2^t 3^t 5^t$, where $t = \lg_{30} n$ (this is only if $n \geq 1$; if $n = 0$, C rejects, exactly as B would do):

$$U \xleftarrow{f,V} 30^{\lg_{30} n};\ \texttt{if}\ \neg f\ \texttt{then reject}.$$

It then removes all 2's and 3's from this product,[6] so that U becomes $2^0 3^0 5^{\lg_{30} n}$,

[5] Some extra care is necessary when $L = 0$ or $R = 0$, in which case the read head is at the leftmost or rightmost input symbol. We omit the details.

[6] To remove all 2's, C divides U by 2 repeatedly ($U \xleftarrow{r,V} \left\lfloor \frac{U}{2} \right\rfloor$), until a non-zero remainder r is returned, which implies there were no 2's in U before the last division. Then the correction

$$U \xleftarrow{\mathtt{t},V} 2U;\ U \xleftarrow{\mathtt{t}} U+1$$

undoes the damage caused by the last division. A similar computation removes all 3's.

which correctly encodes the values of the counters of B *right after the first part of its initialization phase*. Now C starts a faithful simulation of the steps of B.

The current state of B is stored in C's memory. When B attempts to decrease J,

$$J \xleftarrow{f} J - 1,$$

C divides U by 5; if there is no remainder, the division simulated a successful decrement; otherwise, the simulated attempt failed and C restores U's initial value:

$$U \xleftarrow{r,V} \left\lfloor \tfrac{U}{5} \right\rfloor;\ \texttt{if } r = 0 \texttt{ then } f \longleftarrow \texttt{true else}$$
$$\{f \longleftarrow \texttt{false};\ U \xleftarrow{\texttt{t},V} 5U;\ \texttt{repeat } r \texttt{ times: } U \xleftarrow{\texttt{t}} U + 1\}.$$

The attempts to decrease L or R are handled similarly, with 2 or 3 instead of 5.

Attempts of B to increase its counters are simulated by appropriate multiplications of U. The only subtlety involves failure during increment attempts: how does the simulation make sure that *an attempt of B to increase a counter fails* iff *the corresponding attempt of C to multiply U fails?*[7] The crucial observation (from last section) is that, since we are after the first part of B's initialization phase, *every such attempt of B is successful and produces a value at or below t*. Hence, the value of U at any point is at most $2^t 3^t 5^t$, its original value. Therefore, when C multiplies U to simulate a counter increment, it knows in advance the multiplication cannot fail. Overall, B's atomic operation

$$J \xleftarrow{\texttt{t}} J + 1 \qquad \text{is simulated by} \qquad U \xleftarrow{\texttt{t},V} 5U,$$

and similarly for L and R.

The final machine. M' is a 2DFA_2^{u} that, on input n, simulates C on input $n+1$ mimicking the counter operations by head moves (cf. Lemma 4.1).

This concludes the definitions of all four machines of our reduction. It should be clear that A, B, and C are all good, so that M' is good, as well. Moreover, a description z' of M' can clearly be computed out of the description z of M.

6 Conclusion

We have proved that one extra head allows non-recursively greater succinctness in describing languages of **L** by multihead automata, be they deterministic or nondeterministic, unary or not. It is straightforward to draw similar conclusions for other types of (pebble or counter) automata of comparable power.

[7] Note that this condition is necessary for a successful simulation but is not met in some obvious way. In B all three counters have the same upper bound (B's input), whereas in C their representations have different upper bounds: the base 2, base 3, and base 5 logarithms of the upper bound for U (C's input).

References

[1] Jonathan Goldstine, Martin Kappes, Chandra M. R. Kintala, Hing Leung, Andreas Malcher, and Detlef Wotschke. Descriptional complexity of machines with limited resources. *Journal of Universal Computer Science*, 8(2):193–234, 2002.

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

[3] Juris Hartmanis. On the succinctness of different representations of languages. In *International Colloquium on Automata, Languages, and Programming*, volume 71 of *Lecture Notes in Computer Science*, pages 282–288, 1979.

[4] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley, Reading, MA, 1979.

[5] Martin Kutrib. On the descriptional power of heads, counters, and pebbles. In *Proceedings of the 5th Workshop on the Descriptional Complexity of Formal Systems*, pages 138–149, Budapest, 2003. MTA SZTAKI.

[6] Andreas Malcher. Descriptional complexity of cellular automata and decidability questions. *Journal of Automata, Languages and Combinatorics*, 7(4):549–560, 2002.

[7] Albert R. Meyer and Michael J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory*, pages 188–191, 1971.

[8] Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.

[9] Marvin L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.

[10] Burkhard Monien. Transformational methods and their application to complexity problems. *Acta Informatica*, 6:95–108, 1976.

[11] Burkhard Monien. Corrigenda: Transformational methods and their application to complexity problems. *Acta Informatica*, 8:383–384, 1977.

[12] Burkhard Monien. Two-way multihead automata over a one-letter alphabet. *RAIRO Informatique Théorique/Theoretical Informatics*, 14(1):67–82, 1980.

[13] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, April 1959.

[14] Robert W. Ritchie and Frederick N. Springsteel. Language recognition by marking automata. *Information and Control*, 20:313–330, 1972.

[15] Erik M. Schmidt and Thomas G. Szymanski. Succinctness of descriptions of unambiguous context-free languages. *SIAM Journal of Computing*, 6(3):547–553, 1977.

[16] John C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:198–200, April 1959.

[17] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, Boston, MA, 1996.

[18] Richard E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11:323–340, 1967.

[19] Leslie G. Valiant. A note on the succinctness of descriptions of deterministic languages. *Information and Control*, 32:139–145, 1976.

[20] Hao Wang. A variant of Turing's theory of computing machines. *Journal of the ACM*, 4(1):63–92, January 1957.