International Journal of Foundations of Computer Science

NON-RECURSIVE TRADE-OFFS FOR TWO-WAY MACHINES

CHRISTOS KAPOUTSIS
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139, USA, cak@mit.edu

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

If the machines of some type A have enough resources to (i) solve problems that no machine of type B can solve, and (ii) simulate any unary two-way deterministic finite automaton that has access to a linearly-bounded counter, then typically no recursive function can upper bound the increase in the size of description when a machine of type A is replaced by an equivalent machine of type B.

Keywords: Descriptional complexity; Non-recursive trade-offs; Multi-head two-way finite automata; Multi-counter automata.

1. Introduction

The origins of descriptional complexity of formal machines can be traced back to [19], when Rabin and Scott showed that every one-way nondeterministic finite automaton (1NFA) has a deterministic equivalent (1DFA) that is at most *exponentially* larger. The question whether this exponential blowup in size can be avoided has been one of the first steps in the general direction of investigating the relative succinctness of different types of machines as language descriptors.

A similar but also qualitatively distinct situation emerged several years later, when Stearns [22] studied the relation between one-way deterministic pushdown automata (1DPAs) and 1DFAs. Differently from above, he knew that not every 1DPA has a 1DFA-equivalent. But, similarly as above, he proved that, whenever such equivalents exist, the smallest among them are at most *triple-exponentially* larger than the 1DPA.

This naturally lead to the corresponding question for one-way nondeterministic pushdown automata (1NPAs): in the case where a 1NPA has 1DFA-equivalents, what is an upper bound for the size of the smallest among them? The answer was again qualitatively new, by Meyer and Fischer [13], who showed that every such bound grows (as a function of the size of the 1NPA) faster than any computable function. Put another way, among the cases where it is possible to convert a 1NPA into a

1DFA, the trade-off in the size of description is in general *non-recursive*. Several refinements of this result followed [23, 20].

In an important development, Hartmanis [4] later explained that the recursiveness of the trade-off from a type of machines A to a not-as-powerful type of machines B typically implies the recognizability (semi-decidability) of the corresponding *inadequacy problem*: "given a machine of type A, check that it has no B-equivalents". This greatly simplified the proofs of [13, 23, 20], while it nicely revealed the connections of the entire discussion to Gödel's theorem that the addition of an extra axiom to a formal system typically results in non-recursively shorter proofs for some of its theorems [5].

Today many refinements of the above results are known and non-recursive trade-offs have emerged in numerous other comparisons between different types of machines. Comprehensive surveys can be found in [3, 12].

1.1. This Study

In a remark in [7], Hartmanis and Baker showed that *a non-recursive trade-off can occur even when an optimal algorithm replaces a near-optimal one.*[a] For example, converting an $n^{2+\epsilon}$-space deterministic Turing machine (DTM) into one that uses only n^2-space involves a non-recursive blowup in the size of description. In the pattern of [4], they derived this observation from the unrecognizability of the inadequacy problem from near-optimal to optimal machines (from $n^{2+\epsilon}$-space to n^2-space DTMs), which in turn was shown to be a consequence of the fact that the near-optimal complexity class is strictly larger than the optimal one (some $n^{2+\epsilon}$-space DTMs have no n^2-space equivalent).

In the present study we refine that argument. We prove a general theorem that directly shows the non-recursiveness of the trade-off in numerous conversions between machines of different power. In loose terms, our theorem states the following:

If two types of machines A *and* B *are such that*

(1) *some machine of type* A *has no equivalent machine of type* B*, and*

(2) *a machine of type* A *has enough resources to simulate a unary two-way deterministic finite automaton which has access to a linearly-bounded counter,*

then the trade-off from machines of type A *to machines of type* B *is non-recursive.*

For example, for the previous remark on space, we argue that, since $n^2 = o(n^{2+\epsilon})$, there exist $n^{2+\epsilon}$-space DTMs with no n^2-space equivalent, so that condition (1) is clearly true; that (2) is also true follows from the easy observation that any $\Omega(\lg n)$ amount of space suffices for the simulation of a linearly-bounded counter.

The most characteristic applications concern the successive levels of hierarchies of two-way multi*pointer*[b] automata. For example, the following trade-offs are non-

[a] The reader is referred to [7, 5, 6] for a quite interesting discussion of the implications that this might have to our search for optimal algorithms.

[b] By 'pointer' we mean any of the following: a linearly-bounded counter, a *blind* read-only head (i.e., a head that cannot distinguish among different input symbols, although it *can* distinguish between an input symbol and an end-marker), an ordinary read-only head, a *sensing* read-only head (i.e., one that can sense which of the other heads are at the same cell as itself), or a pebble.

recursive:[c]

- from $k+1$ to k counters, on linearly-bounded two-way deterministic counter automata (unary or not) [18],
- from $k+1$ to k heads, on two-way multi-head finite automata (deterministic or not, unary or not) [16, 17, 18],
- from $k+1$ to k heads, on two-way multi-head pushdown automata (deterministic or nondeterministic) [9],

for all k. Sometimes, we can only be as refined as the hierarchy is known to be:

- from $k+2$ to k registers, on linearly-bounded register machines (deterministic or nondeterministic) [18],
- from $k+2$ to k counters, on linearly-bounded two-way nondeterministic counter automata (unary or not) [18],

for all k. Similarly, the trade-off is non-recursive

- from 3 to 2 heads, on a *simple*[d] two-way deterministic finite automaton [1]; it remains non-recursive even when we start from a 2-head two-way deterministic finite automaton, or from a 1-head two-way deterministic pushdown automaton [1].

Finally, we can even conclude the non-recursiveness of the trade-off, for all $k \geq 2$:

- from $k+1$ to k work-tape symbols, on TMs (deterministic or not) that, on every input of length n, use no more than $\lg n$ work-tape cells [21] (even if the starting TM has unary input alphabet, but then only for sufficiently large k).

Other conversions between machines of different power can be treated similarly.

Returning to the statement of the theorem above, we warn that it is, in fact, incomplete. Additional conditions have to be met, concerning A and B, their descriptions, and how 'size' is measured. However, in most interesting cases these conditions are trivially satisfied (in the above examples they are), so that listing them here would be a distraction. The complete list is contained in the formal statement of the theorem in Section 3.

1.2. Outline

The next section describes the framework of this study in more detail. Section 3 states and proves the theorem, except for an important lemma, which is proved in Section 5, after some preparation in Section 4. The discussion is abstract enough to also cover cases where A or B denote types of language descriptors other than machines (e.g., regular expressions, grammars). For a more concrete discussion, see [10]. (There, a special case of the theorem is proved, namely the case concerning the trade-off from $k+1$ to k heads on two-way multi-head finite automata.)

2. Preliminaries

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^*.

[c] In each case, the reference indicates where Condition (1) of the theorem has been established. For Condition (2), it is always easy to see that it is also satisfied.

[d] In a *simple* multi-head automaton, every input head after the first one is *blind* (cf. Footnote b).

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 (possibly by looping) all $x \in \Pi_{\text{no}}$. 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 $x \in \Pi_{\text{yes}} \cup \Pi_{\text{no}}$, 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 clearly Π' is unrecognizable, as well.

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 $0 \leq l \leq \infty$ of 'sufficiently long', $\Pi_{\text{yes}} = \{x \in \Gamma^* \mid \text{length}(x) \geq l\}$, then we say Π *obeys a threshold* (note that then Π_{yes} is empty iff this threshold is infinite); a machine that solves Π is similarly said to obey the same threshold.

2.1. Descriptional Systems

A *descriptional system* over the alphabets Σ and Γ is any set $D \subseteq \Sigma^*$ of *names* (or *descriptors*), along with two total functions $(\cdot)_D$ and $|\cdot|_D$, mapping every name $d \in D$ to its *language* $(d)_D \subseteq \Gamma^*$ and its *size* $|d|_D \in \mathbb{N}$, respectively.

For a standard example, fix a binary encoding of all 1DFAs with input alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, say. This induces the descriptional system over $\{0, 1\}$ and $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ that contains all encoding strings as names and maps each of them to the language accepted by the corresponding 1DFA (as its language) and to the number of states in that 1DFA (as its size). (Alternatively, the size of a name could just be its length.)

A system D is *decidable* if the membership problem for its names is decidable. That is, if there exists a TM U_D that always halts and is such that:

$$\text{for all } d \in D \text{ and } w \in \Gamma^*: \qquad U_D(d, w) \text{ accepts} \iff w \in (d)_D.$$

Thus, the system of the previous example is clearly decidable, whereas a system containing binary encodings of TMs would be undecidable.

In order to be able to compare two descriptional systems D and E in terms of their relative succinctness, we require that they are *comparable*, in the sense that (i) they are defined over the same alphabets, and that (ii) their $(\cdot)$ and $|\cdot|$ mappings agree on all common names,[e]

$$\text{for all } z \in D \cap E: \quad (z)_D = (z)_E \quad \text{and} \quad |z|_D = |z|_E,$$

so that subscripts can be dropped: for all $z \in D \cup E$, (z) and $|z|$ are unambiguous. For such systems, the comparison of E against D involves two natural notions:

I. For a name $e \in E$, there *may* or *may not* exist a name in D that maps to the same language. In the latter case, we say that D *is inadequate for describing the language of* e and, accordingly, we call the associated computational problem, "given an $e \in E$, check that no $d \in D$ maps to (e)", the *inadequacy problem from*

[e]It is only for simplicity that we require the agreement for $|\cdot|$; we do not actually need it.

E *to* D. Formally, this is the promise problem $\mathsf{I} = (\mathsf{I}_{\text{yes}}, \mathsf{I}_{\text{no}})$, with:

$$\mathsf{I}_{\text{yes}} = \{e \in E \mid (d) \neq (e), \text{ for all } d \in D\},$$
$$\mathsf{I}_{\text{no}} = \{e \in E \mid (d) = (e), \text{ for some } d \in D\}.$$

Notice that e is promised to be in E, so that solving I does not require checking membership in E (which might be hard, even impossible).

II. When a name $e \in E$ does have *equivalent* names in D (i.e., names mapping to (e)), we naturally ask how larger than e the smallest of these D-equivalents are. The typical way to answer this question is with a function $\mathsf{f} : \mathbb{N} \to \mathbb{N}$ that upper bounds this increase in size, in terms of the size of e. Namely, f is such that

> for all $s \in \mathbb{N}$ and for all $e \in E$ of size s: if D contains names that are equivalent to e, then at least one of these names is of size at most $\mathsf{f}(s)$.

We say that f *upper bounds the trade-off* (for the conversion) *from* E *to* D. When a computable such upper bound exists, we say *the trade-off from* E *to* D *is recursive*.

As first noted by Hartmanis [4], discussions (I), (II) are not unrelated: unrecognizability of the inadequacy problem typically implies the trade-off is non-recursive.

Lemma 1 (Hartmanis) *Suppose* D, E *are two* comparable *descriptional systems over alphabets* Σ *and* Γ, *and that the following conditions are met:*

(H_1) *both* D *and* E *are decidable,*

(H_2) *for every* $e \in E$, *we can effectively compute its size* $|e|$, *and*

(H_3) *there is a halting* TM *that, on input* $s \in \mathbb{N}$, *produces a list* $Z \subseteq \Sigma^*$ *such that*

- (i) *the non-*D *names can be recognized in* Z: $(Z \cap \overline{D}, Z \cap D)$ *is recognizable.*
- (ii) *the languages of the* D*-names in* Z *cover all and only those languages over* Γ *that are supported by a name in* D *of size at most* s:

$$\{(z) \mid z \in Z \cap D\} = \{(d) \mid d \in D \ \ \& \ \ |d| \leq s\}.$$

Then, recursiveness of the trade-off from E *to* D *implies the corresponding inadequacy problem is recognizable.*

Before giving the proof, let us remark how mild conditions (H_1)–(H_3) are. For most interesting cases, the first two of them are trivially true and (H_3) is satisfied via the TM that simply lists all names in D that have size $\leq s$ (so that the problem of (i) is trivially decidable and the two sets of (ii) trivially identical). Having (H_3) as complicated simply covers some special cases (e.g., comparing general to unambiguous context-free grammars [5, Example 2]).

Proof. Suppose D, E are as in the statement and f is a computable upper bound for the trade-off from E to D. To check that a given $e \in E$ has no D-equivalents, we first compute $s = \mathsf{f}(|e|)$ (by (H_2) and since f is computable) and then run the TM guaranteed by (H_3) on s, to produce a (finite, since the TM is halting) list of names $Z = \{z_1, z_2, \ldots, z_k\}$. At this moment, we know (by the selection of f and (H_3ii)) we should accept iff every D-name in Z maps to a language different from (e). Equivalently, we should accept iff: for every $z \in Z$, *either* z is not a D-name *or* z is a D-name and (z), (e) differ at one or more $w \in \Gamma^*$.

In order to check this, we start simulating, in two parallel threads: [I] the recognizer guaranteed by (H_3i) on each of $z_1, z_2, \ldots, z_k$ in parallel, and [II] for all $w \in \Gamma^*$: the machines U_E and U_D (guaranteed by (H_1)) respectively on (e, w) and on each of $(z_1, w), (z_2, w), \ldots, (z_k, w)$. Whenever a $z \in Z$ is accepted in thread I, we cross it off the list; whenever a $z \in Z$ is found to disagree with e on some w in thread II, it is crossed off the list, as well; if the list ever gets empty, we accept.

Clearly, every string in Z that is not a D-name, will eventually be crossed off, in thread I; similarly, each D-name that is inequivalent to e will also be eventually removed, in thread II; moreover, neither thread can delete a D-name that is equivalent to e. Hence, the list will eventually get empty iff e had no D-equivalent in the original list Z; which is true iff e has no D-equivalent at all. □

2.2. Multi-Counter Automata

A *deterministic automaton with k counters*[f] (DCA_k) consists of a finite state control and k counters, each of which can store a nonnegative integer. One of the counters is distinguished as *primary*, the rest being referred to as *secondary*. The input to the automaton is a nonnegative upper bound n for the primary counter. The machine 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 is the primary one and it already contains n; an attempt to increase a secondary counter never fails. 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*, and the *language* of the machine is exactly the set of inputs that it accepts. If, for all n, the behavior of the automaton guarantees no secondary counter ever grows larger than n, we say the automaton is (*linearly*) *bounded*.

We will be interested in a special version of the emptiness problem for multi-counter automata. One way to introduce it is to start with the emptiness problem for TMs ("given a description of a TM, check that its language is empty"), which is well known to be unrecognizable [8], and to consider some 'simplifications' to it:

- What happens if, instead of a full-fledged TM, the machine we are given is 'simpler'? Say, a multi-counter automaton? Or just a DCA_2? Clearly, checking emptiness becomes 'simpler', too. Does it also become recognizable?
- What if, in addition, the given DCA_2 is promised to be *bounded*? And *terminating*, too? And to also *obey a threshold*? As the promise gets stronger, checking emptiness again becomes 'simpler'. But does it become recognizable?

[f] In this definition the reader will recognize *the unary version* of the *two-way $(k-1)$-counter machines* of [2, 18] or of other automata from elsewhere; they all describe the natural notion of a *unary two-way deterministic finite automaton that has additional access to $k-1$ counters*. The present model is a bit simpler, in the sense that the input is directly taken to be the upper bound for one of the counters, saving us the redundant (in the unary case) notion of an input tape. (Note that this definition differs from that of [10], where the upper bound is applied to all counters.)

So, the problem we want to define is: "given a description of a DCA_2 that is promised to be bounded and terminating and to obey a threshold, check that its language is empty." In formal dialect, $\mathsf{E} = (\mathsf{E}_{\text{yes}}, \mathsf{E}_{\text{no}})$, where

$$\mathsf{E}_{\text{yes}} = \{z \in \langle \text{DCA}_2' \rangle \mid (z) = \emptyset\} \qquad \text{and} \qquad \mathsf{E}_{\text{no}} = \{z \in \langle \text{DCA}_2' \rangle \mid (z) \neq \emptyset\},$$

$\langle \text{DCA}_2' \rangle$ stands for the set of descriptions (under a fixed encoding) of all terminating, bounded DCA_2s that obey a threshold, and (z) denotes the language of the machine described by z. Interestingly, although not surprisingly, even for such a weak automaton and under such a strong promise, emptiness remains unrecognizable:[g]

Lemma 2 E *is unrecognizable.*

We use this fact in the next section, but defer proving it until Section 5. In between, Section 4 discusses the capabilities of multi-counter automata.

3. The Main Theorem

We are now ready to state and prove the main theorem.

Theorem 1 *Suppose D, E are two* comparable *descriptional systems that satisfy conditions* (H_1)–(H_3) *of Lemma* 1*. If they also satisfy the following:*

(C_1) *there exists a name $e_0 \in E$ that has no equivalent in D,*

(C_2) *given a description z of a* terminating, bounded DCA_2 *that* obeys a threshold, *we can effectively construct a name $e_z \in E$ such that*

$$(e_z) \;=\; (e_0) \;\cup\; \{w \in \Gamma^* \mid \text{length}(w) \in (z)\}, \tag{1}$$

(C_3) *every* co-finite *language has a name in D that maps to it,*

then the trade-off from E to D is non-recursive.

Before proving the theorem, we discuss how mild conditions (C_1)–(C_3) really are. Since every co-finite language is regular, (C_3) is trivially satisfied whenever the names in D describe machines that have some kind of finite state control.

The second condition essentially says that the machines described by E have enough resources to simulate a bounded DCA_2. Because then, from a given z, we can always construct the description e_z of the E-machine that does the following:

> on input $w \in \Gamma^*$: first simulate on length(w) the DCA_2 described by z; if this accepts, then halt and accept; otherwise, simulate on w the machine described by e_0 and accept, reject, or loop accordingly.

and which obviously satisfies (1) (note the importance of the promise that z describes a DCA_2 that never rejects by looping). Given how weak bounded DCA_2s are, most two-way machines with non-regular capabilities will easily meet (C_2).

The important condition is (C_1), which requires that the machines described by D are *not as powerful as* those described by E; in other words, a separation is needed between the complexity classes that correspond to the two systems.

[g] Note that clearly $\mathsf{E} \in \Pi_1$ and that the proof of Lemma 2 will show E is Π_1-complete. We also remark that, under no promise and after non-trivially modifying the definition of DCA_2s, E is the emptiness problem for 2-register machines, which is well known to be Π_1-complete [14].

Proof. We essentially repeat Hartmanis' argument from [5, Example 4] (see also [11, Theorem 7]). Suppose D, E are as in the statement of the theorem. Since (H_1)–(H_3) are satisfied, Lemma 1 implies that we only need to prove that the inadequacy problem I from E to D is unrecognizable. By Lemma 2, we just need to reduce E to it:

$$\mathsf{E} \leq \mathsf{I}.$$

Given a $z \in \langle \text{DCA}'_2 \rangle$, we simply construct the name $e_z \in E$ guaranteed by conditions (C_1) and (C_2), so that

$$(e_z) \;=\; (e_0) \;\cup\; \{w \in \Gamma^* \mid \text{length}(w) \in (z)\}.$$

If $z \in \mathsf{E}_{\text{yes}}$, then the language of z is empty, so that $(e_z) = (e_0)$ and e_z has no D-equivalent (because e_0 does not); hence $e_z \in \mathsf{I}_{\text{yes}}$. On the other hand, if $z \in \mathsf{E}_{\text{no}}$, then the language of z contains all sufficiently large $w \in \Gamma^*$, so that (e_z) is co-finite and has D-equivalents (by (C_3)); hence $e_z \in \mathsf{I}_{\text{no}}$. This concludes the proof.[h] □

4. Programming Counters

In order to present the capabilities of multi-counter automata, 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

$$\texttt{repeat}\;\; X \xleftarrow{f} X - 1 \;\; \texttt{until}\; \neg f. \tag{2}$$

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

$$(X, Y) \xleftarrow{f} (Y, 0),$$

and describe it by a line similar to (2). Note that, if X is the primary counter and $Y > n$, then 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. So, X's original contents are always lost, but this never happens to the original contents of Y.

Changing how fast X increases as Y decreases, we can multiply/divide Y into X by any constant $a \in \mathbb{N}$. We denote these operations by

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

[h] Note the slightly stronger fact: I remains unrecognizable even under the promise that the given $e \in E$ either has no D-equivalent or its language is co-finite. In addition, the promise that the given DCA'_2 obeys a threshold can be slightly relaxed: we only need to know that its language is either empty or co-finite.

where, in the second operation, we also find the remainder and return it in r. As before, if X is the primary counter and $aY > n$ (respectively, $\lfloor Y/a \rfloor > n$) then one of the attempts to increase X will fail; we then restore the original value of Y returning X to 0, and set flag f to false.

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:[i]

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

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

$$Y \xleftarrow{f,X} aY \qquad \text{and} \qquad Y \xleftarrow{f,r,X} \left\lfloor \tfrac{Y}{a} \right\rfloor. \tag{3}$$

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

$$\begin{array}{l} X \longleftarrow 0;\ X \xleftarrow{f} X + 1; \\ \texttt{if } f \texttt{ then } \{Y \longleftarrow 0;\ Y \xleftarrow{\texttt{t}} Y + 1;\ \texttt{repeat } Y \xleftarrow{g,X} aY \texttt{ until } \neg g\} \end{array} \tag{4}$$

(note that this fails iff $n = 0$). We denote (4) 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 (4) to also count (in Z) the number of iterations performed. This gives us a way 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 (3) so that the success of the operation depends on the *contents* of X (as opposed to its upper bound n):

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

meaning that, using Z as auxiliary and without affecting X: if $aY \leq X$, then Y is set to aY; otherwise, Y is unaffected.[j] Then, the following variant of (4)

$$\begin{array}{l} X \xleftarrow{f} X - 1;\ \texttt{if } f \texttt{ then } \{X \xleftarrow{\texttt{t}} X + 1; \\ \qquad\qquad Y \longleftarrow 0;\ Y \xleftarrow{\texttt{t}} Y + 1;\ \texttt{repeat } Y \xleftarrow{g,X,Z} aY \texttt{ until } \neg g\} \end{array}$$

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

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

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

[j] To implement this, we first set Z to 0. Then, we repeatedly decrease Y, increase Z, and decrease X by a. *If X becomes 0 before Y*, then $aY > X$ and the operation should fail: we restore the original values of Y and X by repeatedly decreasing Z, increasing Y, and increasing X by a, until Z becomes 0. *Otherwise*, $aY \leq X$ and the operation will succeed: we copy the correct value to Y and restore the value of X by repeatedly decreasing Z and increasing each of Y, X by a, until Z becomes 0. Note that *if originally $Y, Z \leq X$, then at no point during the operation does any of the counters assume a value greater than the original value of X.*

It is important to note that, *if originally $Y, Z \leq X$, then during this operation no counter ever assumes a value greater than the original value of X* (cf. Footnote j).

5. Proof of Lemma 2

We now prove that E is unrecognizable. We do this by a reduction from the complement of the halting problem, which is known to be unrecognizable [8]:

$$\overline{\mathsf{HALTING}} \leq \mathsf{E},$$

where $\overline{\mathsf{HALTING}} = \{z \in \{0,1\}^* \mid z \text{ encodes a TM that loops on } z\}$. That is, we give an algorithm that, on input a description z of a TM M produces a description z' of a terminating, bounded, threshold-obeying DCA_2 M', such that

$$M \text{ loops on } z \implies (z') = \emptyset \qquad \text{and} \qquad M \text{ halts on } z \implies (z') \neq \emptyset. \tag{5}$$

In describing this algorithm, we will be calling a machine (TM or DCA_k) *good*, if it is terminating, bounded (for DCA_ks), obeys a threshold, and its language satisfies (5) when it replaces (z'). Thus, for example, M' will be good.

On its way to z', the algorithm will construct descriptions of two other machines: a description z_A of a TM A, and a description z_B of a DCA_3 B. In the sequence M, A, B, M' each machine after M will be defined in terms of the previous one and will be good. Our constructions use the ideas of [24] and [14] (also found in [15, 8]).

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 on the $\sqcup$ next to the leftmost 0 (or any $\sqcup$, if $n = 0$). It then computes:

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

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; and without ever visiting any $\sqcup$ that lies beyond the two that originally delimit the input. 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). Also note that, by the selection of the encoding scheme for M's computation histories, if A accepts an input 0^n, it necessarily accepts all longer inputs as well.

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.

To explain B's behavior, let J, L, R be its three counters. J is primary and helps 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	h	r_0	r_1	r_2	r_3	r_4	r_5	$\cdots$
$\cdots$	$\sqcup$	$\sqcup$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\sqcup$	$\sqcup$	$\cdots$
						$\uparrow$							

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

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

(note that the two sums are finite exactly because $\sqcup$ maps to 0). The values l and r are kept in L and R, while h is kept in a register H in B's finite memory.

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 $n = 0$, B rejects):

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

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

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

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

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

Hence L, H, R correctly represent A's starting tape configuration on input 0^m:

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

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

The automaton remembers in its finite memory the current state of A as well as the code of the currently read symbol (in H). If s is the code of the new symbol to be written on the tape, B computes

$$L \xleftarrow{\mathtt{t},J} 5L;\ \texttt{repeat}\ s\ \texttt{times:}\ L \xleftarrow{\mathtt{t}} L + 1;\ R \xleftarrow{\mathtt{t},r_0,J} \left\lfloor \tfrac{R}{5} \right\rfloor;\ H \longleftarrow r_0$$

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

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

to simulate writing this symbol and moving to the left.

[k] Any mapping that maps symbol $\sqcup$ to code 0 and symbol $\mathtt{0}$ to code 4 will do.

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 operations 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 final machine. M' is a DCA_2 that simulates the behavior of B. If U, V are its two counters, then U is primary and helps performing operations on V, while V encodes the contents of the counters of B: whenever J, L, R, contain j, l, r respectively, V contains $2^j 3^l 5^r$.

The automaton starts by computing into V the product $30^t = 2^t 3^t 5^t$, where $t = \lg_{30} n$ (this is only if $n \geq 1$; if $n = 0$, M' rejects, exactly as B would do):

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

It then removes all 3s and 5s from this product,[l] so that V becomes $2^{\lg_{30} n} 3^0 5^0$, which correctly encodes the values of the counters of B *right after the first part of its initialization phase*. Now M' starts a faithful simulation of the steps of B.

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

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

M' divides V by 2; if there is no remainder, the division simulated a successful decrement; otherwise, the simulated attempt failed and M' restores V's initial value:

$$V \xleftarrow{\mathtt{t},r,U} \left\lfloor \tfrac{V}{2} \right\rfloor;\ \texttt{if}\ r = 0\ \texttt{then}\ f \longleftarrow \texttt{true else}$$
$$\{f \longleftarrow \texttt{false};\ V \xleftarrow{\mathtt{t},U} 2V;\ \texttt{repeat}\ r\ \texttt{times:}\ V \xleftarrow{\mathtt{t}} V + 1\}.$$

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

Attempts of B to increase its counters are of course simulated by appropriate multiplications of V. 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 M' to multiply V fails?*[m] The crucial observation

[l] To remove all 3s, M' divides V by 3 repeatedly ($V \xleftarrow{\mathtt{t},r,U} \left\lfloor \frac{V}{3} \right\rfloor$), until a non-zero remainder r is returned, which implies there were no 3s in V before the last division. Then the correction

$$V \xleftarrow{\mathtt{t},U} 3V;\ \texttt{repeat}\ r\ \texttt{times:}\ V \xleftarrow{\mathtt{t}} V + 1$$

undoes the damage caused by the last division. A similar computation removes all 5s.

[m] Note that this condition is necessary for a successful simulation but is not met in some obvious way. In B the upper bound for J is always the same (B's input), whereas in M' the upper bound for its representation is the base 2 logarithm of a value that depends (on M''s input and) on the values of the other two counters. Similarly, in B counter L is unrestricted, whereas in M' its representation is bounded by a value that depends on the other two counters; similarly for R.

(from last section) is that, since we are after the first part of B's initialization phase, *no counter of B ever assumes a value greater than $t = \lg_{30} n$*. This immediately implies that, after the initialization phase of M', *no counter of M' ever assumes a value greater than $2^t 3^t 5^t$*. Now, since $t < n$ and $2^t 3^t 5^t \leq n$, we conclude that both (i) all increment attempts of B are successful, and (ii) all corresponding multiplication attempts of M' are successful, too. Hence, the equivalence above is satisfied, in a trivial way. Put another way, when M' multiplies V to simulate a counter increment in B, it knows in advance the increment is a successful one and the multiplication will not fail. Overall, B's atomic operation

$$J \xleftarrow{\mathtt{t}} J + 1 \qquad \text{is simulated by} \qquad V \xleftarrow{\mathtt{t},U} 2V,$$

and similarly for L and R.

As a final remark, we note the immediate by-product of our last argument: Since V clearly never exceeds n during the initialization phase of M' and it also never exceeds n during the simulation of B, it follows that M' is bounded.

This concludes the definitions of all three machines in our reduction. It should be clear that M' is good and that a description z' of it can be computed out of z.

6. Conclusion

Using old ideas [24, 14], we showed the unrecognizability of the emptiness problem for DCA_2s that are promised to be bounded, always terminate, and obey a threshold. We then combined this with the idea of [7] to show that, if machines A have the resources to simulate DCA_2s of the particular kind and can also solve problems that machines B cannot, then typically the trade-off from A to B is non-recursive. Applying the theorem, we derived such trade-offs in many conversions.

We do not know if the emptiness problem of Section 2.2 remains unrecognizable even when the underlying machine is a 2-*register automaton* [14] (that is, a DCA_2 that starts with n in its primary counter and where increments of that counter never fail). If it is, then our main theorem can be made slightly stronger.

Acknowledgments

I would like to thank Oscar Ibarra and Albert Meyer for discussions that revealed the generality behind the argument of [10].

References

1. P. Ďuriš and Z. Galil, "Fooling a two-way automaton *or* one pushdown store is better than one counter for two-way machines," *Theoret. Comput. Sci.* **21** (1982) 39–53.
2. M. J. Fischer, A. R. Meyer, and A. L. Rosenberg, "Counter machines and counter languages," *Mathematical Syst. Theory* **3** (1968) 265–283.
3. J. Goldstine, M. Kappes, C. M. R. Kintala, H. Leung, A. Malcher, and D. Wotschke, "Descriptional complexity of machines with limited resources," *J. UCS* **8** (2002) 193–234.

4. J. Hartmanis, "On the succinctness of different representations of languages," *SIAM J. Comput.* **9** (1980) 114–120.
5. J. Hartmanis, "On Gödel speed-up and succinctness of language representations," *Theoret. Comput. Sci.* **26** (1983) 335–342.
6. J. Hartmanis, "On the importance of being Π_2-hard," *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS* **37** (1989) 117–127.
7. J. Hartmanis and T. P. Baker, "Relative succinctness of representations of languages and separation of complexity classes," *Proc. International Symposium on Mathematical Foundations of Computer Science*, 1979, pp. 70–88.
8. J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation* (Addison-Wesley, Reading, MA, 1979).
9. O. H. Ibarra, "On two-way multihead automata," *J. Comput. System Sci.* **7** (1973) 28–36.
10. C. Kapoutsis, "From $k+1$ to k heads the descriptive trade-off is non-recursive," *Proc. Workshop on the Descriptional Complexity of Formal Systems*, 2004, pp. 213–224.
11. M. Kutrib, "On the descriptional power of heads, counters, and pebbles," *Proc. Workshop on the Descriptional Complexity of Formal Systems*, 2003, pp. 138–149.
12. M. Kutrib, "The phenomenon of non-recursive trade-offs," *Proc. Workshop on the Descriptional Complexity of Formal Systems*, 2004, pp. 83–97.
13. A. R. Meyer and M. J. Fischer, "Economy of description by automata, grammars, and formal systems," *Proc. Symposium on Switching and Automata Theory*, 1971, pp. 188–191.
14. M. L. Minsky, "Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines," *Ann. of Math.* **74** (1961) 437–455.
15. M. L. Minsky, *Computation: finite and infinite machines* (Prentice-Hall, Englewood Cliffs, NJ, 1967).
16. B. Monien, "Transformational methods and their application to complexity problems," *Acta Inform.* **6** (1976) 95–108.
17. B. Monien, "Corrigenda: Transformational methods and their application to complexity problems," *Acta Inform.* **8** (1977) 383–384.
18. B. Monien, "Two-way multihead automata over a one-letter alphabet," *Theor. Inform. Appl.* **14** (1980) 67–82.
19. M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development* **3** (1959) 114–125.
20. E. M. Schmidt and T. G. Szymanski, "Succinctness of descriptions of unambiguous context-free languages," *SIAM J. Comput.* **6** (1977) 547–553.
21. J. I. Seiferas, "Relating refined space complexity classes," *J. Comput. System Sci.* **14** (1977) 100–129.
22. R. E. Stearns, "A regularity test for pushdown machines," *Inform. and Comput.* **11** (1967) 323–340.
23. L. G. Valiant, "A note on the succinctness of descriptions of deterministic languages," *Inform. and Comput.* **32** (1976) 139–145.
24. H. Wang, "A variant of Turing's theory of computing machines," *J. ACM* **4** (1957) 63–92.