Keyboards as a New Model of Computation

Yoan Géran
ENS Paris-Saclay, France

Bastien Laboureix
ENS Paris-Saclay, France

Corto Mascle
ENS Paris-Saclay, France

Valentin D. Richard
ENS Paris-Saclay, France

Abstract

We introduce a new formalisation of language computation, called keyboards. We consider a set of atomic operations (writing a letter, erasing a letter, going to the right or to the left) and we define a keyboard as a set of finite sequences of such operations, called keys. The generated language is the set of words obtained by applying some non-empty sequence of those keys. Unlike classical models of computation, every key can be applied anytime. We define various classes of languages based on different sets of atomic operations, and compare their expressive powers. We also compare them to rational, context-free and context-sensitive languages. We obtain a strict hierarchy of classes, whose expressiveness is orthogonal to the one of the aforementioned classical models. We also study closure properties of those classes, as well as fundamental complexity problems on keyboards.

2012 ACM Subject Classification Theory of computation

Keywords and phrases formal languages, models of computation, automata theory

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.49

Related Version *Full version in both French and English*: `https://arxiv.org/abs/2102.10182`

Acknowledgements We want to thank Pierre Béaur, Lucas Buéri, Jean-Baptiste Daval, Paul Gastin, Colin Geniet, Valentin Maestracci and Clément Théron for their helpful advice and feedback.

1 Introduction

We present a new formalisation of languages, called keyboards. A keyboard K is a finite set of keys, which are finite sequences of atomic operations, such as writing or erasing a letter, going one position to the right or to the left... The language of K is the set of words obtained by applying a sequence of its keys on an initially empty writing space. The idea comes from the image of a malfunctioning keyboard, in which a key does a sequence of operations instead of just one (for instance the key 'a' writes b then c, and then goes to the left).

Studying the set generated by a set of algebraic operations is far from new: many works exist on the sets generated by a subset of elements of an algebraic structure, for instance in the context of semigroup and group theory [2, 7], of matrix monoids [1, 8] or the theory of codes [3]. There is however, to the best of the author's knowledge, no previous work on a model resembling the one presented here.

The atomic operations we use in this paper are the base of other models of computation, such as forgetting automata and erasing automata [4, 5, 9]. The use of those operations was originally to simulate some analysis strategies in linguistics. As a first study of the model, we chose the actions of the operations (backspace and arrows) to behave like an actual keyboard in a text editor.

We can define various classes of languages based on the set of atomic operations we consider, and compare their expressive powers between them, and to well-known classes of languages. We obtain a strict hierarchy of classes, with a wide range of expressiveness and difficulty of comprehension. The expressiveness of keyboards seems to be overall orthogonal to the ones of classical models of computation, which we explain by two key differences with the latter.

First, keyboards are blind and memoryless, in that they do not have states and cannot read the tape at any point. Second, because of this weakness, we can allow operations such as moving in the word or erasing letters without blowing up their expressive power too much.

The main interests of keyboards are: 1. to obtain many deep and complex mathematical questions from a deceptively simple model, and 2. that their expressiveness is very different from the ones of classical models. A language that is simple to express with a keyboard may be more complicated for automata, and vice versa. This paper is meant as a first step in the study and comprehension of keyboards and their languages.

The paper is organised as follows. In Sections 2 and 3 we establish notations and basic definitions. Section 4 and 5 are dedicated to building properties and tools necessary to the study of keyboards. In Section 6 we dive into the specific properties of each keyboard class, and prove the inclusions of some of them in regular, context-free and context-sensitive languages. Then in Section 7, we study the inclusions between those classes, in particular showing that they are all different. Some complexity results are given in Section 8. Finally, in Section 9 we show that keyboard classes are not stable by union or intersection, and that some (but not all) of them are stable by mirror.

We do not provide full proofs, but only sketch of proofs for some results. The technical details can be found in the full version.

2 Preliminaries

Given a finite alphabet A, we note A^* the set of finite words over A and A^+ for the set of non-empty ones. Given a word $w = a_1 \cdots a_n \in A^*$, we write $|w|$ for its length and, for all $a \in A$, $|w|_a$ the number of occurrences of a in w. For all $1 \leq i, j \leq n$ we use the notation $w[i]$ for the i^{th} letter of w (i.e. a_i) and $w[i, j]$ for its factor $a_i \cdots a_j$ (and ε if $j < i$). We denote the mirror of w by $\tilde{w} = a_n \cdots a_1$.

We write $\mathrm{Pref}(w)$ for the set of prefixes of w, $\mathrm{Suff}(w)$ for its set of suffixes, $\mathrm{Fact}(w)$ for its set of factors and $\mathrm{Sub}(w)$ for its set of subwords.

We represent a finite automaton A as a tuple $(Q, \Delta, \mathrm{Init}, \mathrm{Fin})$ with Q a finite set of states, $\Delta : Q \times A \to 2^Q$ a transition function, and $\mathrm{Init}, \mathrm{Fin} \subseteq Q$ sets of initial and final states.

We represent a pushdown automaton on A as a tuple $(Q, \Gamma, \bot, \Delta, \mathrm{Init}, \mathrm{Fin})$ with

- Q a finite set of states ;
- Γ a finite stack alphabet ;
- $\bot \in \Gamma$ an initial stack symbol ;
- $\Delta \colon Q \times A \times (\Gamma \cup \{-\})^2 \to 2^Q$ a transition function ;
- Init and Fin sets of initial an final states.

We accept a word on final states with an empty stack. We write transitions as follows:

$$s_1 \xrightarrow[\mathrm{op}_1, \mathrm{op}_2]{a} s_2$$

with

- $\mathrm{op}_1 = \uparrow\gamma$ if we pop $\gamma \in \Gamma$, and $\mathrm{op}_1 = -$ if we do not pop anything.
- $\mathrm{op}_2 = \downarrow\gamma$ if we push $\gamma \in \Gamma$ on the stack, and $\mathrm{op}_2 = -$ if we do not push anything.

We will use ε-transitions in both finite and pushdown automata to simplify some proofs.

For more details and properties of those models, we refer the reader to [6].

3 Definitions

We fix a finite alphabet A and the following special symbols, taken out of A:

The backspace : $\leftarrow$ The left arrow : $\blacktriangleleft$ The right arrow : $\blacktriangleright$

The set of all symbols is $S \triangleq A \cup \{\leftarrow, \blacktriangleleft, \blacktriangleright\}$. An element of S is called an *atomic operation*.

▶ **Definition 1.** *A* configuration *is a pair of words $(u, v) \in A^* \times A^*$. We will use $\mathcal{C}(A)$ to denote the set of configurations over A, and $\langle u|v\rangle$ to denote the configuration (u, v).*

We define the notation $\langle u|v\rangle_i$ as the letter at position i in the configuration with respect to the cursor: $\langle u|v\rangle_i = \widetilde{u}[-i]$ if $i < 0$ and $v[i]$ if $i > 0$.

▶ **Definition 2.** *The* action of an atomic operation $\sigma \in S$ *on a configuration $\langle u|v\rangle$ is written $\langle u|v\rangle \cdot \sigma$ and is defined as follows:*

$$\langle u|v\rangle \cdot a = \langle ua|v\rangle \text{ if } a \in A.$$
$$\langle \varepsilon|v\rangle \cdot \leftarrow = \langle \varepsilon|v\rangle \quad \text{and} \quad \langle u'a|v\rangle \cdot \leftarrow = \langle u'|v\rangle$$
$$\langle \varepsilon|v\rangle \cdot \blacktriangleleft = \langle \varepsilon|v\rangle \quad \text{and} \quad \langle u'a|v\rangle \cdot \blacktriangleleft = \langle u'|av\rangle$$
$$\langle u|\varepsilon\rangle \cdot \blacktriangleright = \langle u|\varepsilon\rangle \quad \text{and} \quad \langle u|av'\rangle \cdot \blacktriangleright = \langle ua|v'\rangle$$

We will sometimes write $\langle u|v\rangle \xrightarrow{\sigma} \langle u'|v'\rangle$ for $\langle u'|v'\rangle = \langle u|v\rangle \cdot \sigma$.

▶ **Example 3.** By applying the following sequence of atomic operations $\leftarrow$, a, $\blacktriangleright$, $\blacktriangleright$, b to the configuration $\langle c|d\rangle$, we obtain the following rewriting derivation:

$$\langle c|d\rangle \xrightarrow{\leftarrow} \langle \varepsilon|d\rangle \xrightarrow{a} \langle a|d\rangle \xrightarrow{\blacktriangleright} \langle ad|\varepsilon\rangle \xrightarrow{\blacktriangleright} \langle ad|\varepsilon\rangle \xrightarrow{b} \langle adb|\varepsilon\rangle.$$

▶ **Definition 4.** *We define other semantics for atomic operations, called* effective semantics. *The difference with the previous ones is that we forbid application of atomic operations without effect (such as backspace when the left word of the configuration is empty). Formally, given $u, v \in A^*, a \in A$ we have:*

$$\langle u|v\rangle \xrightarrow{a}_e \langle ua|v\rangle \qquad \langle u'a|v\rangle \xrightarrow{\leftarrow}_e \langle u'|v\rangle$$
$$\langle u'a|v\rangle \xrightarrow{\blacktriangleleft}_e \langle u'|av\rangle \qquad \langle u|av'\rangle \xrightarrow{\blacktriangleright}_e \langle ua|v'\rangle$$

We also define the operator $\odot$ by $\langle u|v\rangle \odot \sigma = \langle u'|v'\rangle$ if and only if $\langle u|v\rangle \xrightarrow{\sigma}_e \langle u'|v'\rangle$.

▶ **Definition 5.** *A* key *is a sequence of atomic operations, seen as a word on S. We will use $\mathcal{T}(S)$ to denote the set of keys on S (variables $k, t, ...$), or $\mathcal{T}$ if there is no ambiguity.*

▶ **Definition 6.** *The* action of a key *over a configuration is defined inductively as follows:*

$$\begin{cases} \langle u|v\rangle \cdot \varepsilon = \langle u|v\rangle \\ \langle u|v\rangle \cdot (\sigma t) = (\langle u|v\rangle \cdot \sigma) \cdot t \end{cases}$$

We extend the notation $\langle u|v\rangle \xrightarrow{t} \langle u'|v'\rangle$ to keys. We define $\langle u|v\rangle \xrightarrow{t}_e \langle u'|v'\rangle$ and $\langle u|v\rangle \odot t$ analogously.

▶ Remark 7. We will also consider sequences of keys $\tau = t_1 \dots t_n$. The action of τ is obtained by composing the actions of the t_i, hence applying τ has the same effect as applying sequentially $t_1, \dots, t_n$. Note that τ is seen as a word on $\mathcal{T}(S)$ (and not on S), thus $|\tau|$ is n.

▶ **Definition 8.** *The* length of a key t, *written* $|t|$, *is its length as a word on* S. *Further, given* $\sigma \in S$, *we note* $|t|_\sigma$ *the number of occurrences of* σ *in* t. *The size of a configuration* $\langle u|v\rangle$ *is defined as* $|\langle u|v\rangle| = |u| + |v|$.

▶ **Definition 9.** *Two keys* t *and* t' *are* equivalent, *denoted* $t \sim t'$, *if for all* $u, v \in A^*$, $\langle u|v\rangle \cdot t = \langle u|v\rangle \cdot t'$.

▶ **Example 10.** ε is equivalent to $a\leftarrow$ for all $a \in A$, but not to $\blacktriangleright\blacktriangleleft$, as we have $\langle a|\varepsilon\rangle \cdot \blacktriangleright\blacktriangleleft = \langle \varepsilon|a\rangle$ whereas $\langle a|\varepsilon\rangle \cdot \varepsilon = \langle a|\varepsilon\rangle$.

Example 10 illustrates how $\blacktriangleleft$, $\blacktriangleright$ and $\leftarrow$ act differently if one side of the configuration is empty. We will see that these "edge effects" add some expressiveness compared to the effective semantics, but make proofs more complex.

▶ **Definition 11** (Automatic Keyboard)**.** *An automatic keyboard is a finite subset of* $\mathcal{T}(S)$.

▶ **Definition 12.** *An* execution *of an automatic keyboard* K *on a configuration* $c_0 \in \mathcal{C}$ *is a* ***non-empty*** *finite sequence* $\rho = (t_1, c_1), \ldots, (t_{n+1}, c_{n+1}) \in (K \times C)^{n+1}$ $(n \in \mathbb{N})$ *such that*

$$\forall i \in [\![1\,; n+1]\!], c_{i-1} \xrightarrow{t_i} c_i.$$

By default, we take as initial configuration $c_0 = \langle \varepsilon|\varepsilon\rangle$. *We usually write* $c_0 \xrightarrow{\tau} c_{n+1}$ *to mean the execution* $(\tau[1], c_0 \cdot \tau[1]), ..., (\tau[n+1], c_0 \cdot \tau[1, n+1])$.

▶ **Definition 13.** *A word* $w \in A^*$ *is* recognised *by an automatic keyboard* K *if there exist* $u, v \in A^*$ *and an execution* $\langle \varepsilon|\varepsilon\rangle \xrightarrow{\tau} \langle u|v\rangle$ *such that* $w = uv$. *The language* $\mathcal{L}(K)$ *of* K *is the set of words recognised by* K.

We now define keyboards as automatic keyboards to which we added some final keys "with entry", which mark the end of the execution.

▶ **Definition 14** (Keyboard (with entry))**.** *A keyboard* K *on* S *is a pair* (T, F) *of finite sets* $T, F \subset \mathcal{T}(S)$. *We call the elements of* F *the* final keys *of* K *and the elements of* T *its* transient keys.

▶ **Definition 15** (Accepting execution of a keyboard)**.** *Let* $K = (T, F)$ *be a keyboard and* $c_0 = \langle u_0|v_0\rangle$ *an initial configuration. An* accepting execution *of* K *on* c_0 *is a finite sequence* $\rho = (t_1, c_1), \ldots, (t_{n+1}, c_{n+1}) \in (T \times C)^n \cdot (F \times C)$ $(n \in \mathbb{N})$ *such that*

$$\forall i \in [\![1\,; n+1]\!], c_{i-1} \xrightarrow{t_i} c_i.$$

By default, an accepting execution is on the empty configuration $\langle \varepsilon|\varepsilon\rangle$.

▶ **Definition 16.** *A word* $w \in A^*$ *is* recognised *by a keyboard* K *if there exist* $u, v \in A^*$ *and an execution* $\langle \varepsilon|\varepsilon\rangle \xrightarrow{\tau} \langle u|v\rangle$ *such that* $w = uv$. *The language* $\mathcal{L}(K)$ *of* K *is the set of words recognised by* K.

▶ **Example 17.** The keyboard with one transient key `aa` and one final key `a`, recognises sequences of a of odd length.

▶ Remark 18. Let K_a be an automatic keyboard, then $\mathcal{L}(K_a)$ is recognised by the keyboard with entry (K_a, K_a). In all that follows, we will thus see automatic keyboards as a subclass of keyboards.

▶ **Definition 19** (Size of a keyboard). *The size of a keyboard $K = (T, F)$ is defined as*

$$\|K\|_\infty = \max\{|t| \mid t \in T \cup F\}.$$

We may also use another measure $|K|$ of the size of K for complexity purposes:

$$|K| = \sum_{t \in T \cup F} (|t| + 1).$$

▶ **Definition 20** (Minimal keyboard). *A minimal keyboard K is an automatic keyboard without any operation besides writing letters. It can therefore be seen as a finite subset of A^*. We will note* MK *the class of minimal keyboards.*

▶ Remark 21. We construct our keyboard classes through the sets of special operations we allow. Class names are obtained by adding B (for $\leftarrow$), E (for the entry, noted ■), L (for ◀) and A (for ◀ and ▶) to K. We obtain these classes.

MK : $\{\}$	LK : $\{◀\}$	AK : $\{◀, ▶\}$
EK : $\{■\}$	LEK : $\{◀, ■\}$	EAK : $\{◀, ▶, ■\}$
BK : $\{\leftarrow\}$	BLK : $\{◀, \leftarrow\}$	BAK : $\{◀, ▶, \leftarrow\}$
BEK : $\{\leftarrow, ■\}$	BLEK : $\{◀, \leftarrow, ■\}$	BEAK : $\{◀, ▶, \leftarrow, ■\}$

▶ Remark 22. We do not consider classes with ▶ without ◀ because, without the ◀ operator, we can only reach configurations of the form $\langle u|\varepsilon\rangle$ and thus ▶ never has any effect.

▶ Remark 23. We use the class names above to designate both keyboard classes and language classes. For instance, we will write that L is in AK if there exists a keyboard $K \in$ AK such that $L = \mathcal{L}(K)$.

4 General properties

In this section, we establish some properties on keyboard. Although most of them are quite intuitive, we take the time to be as formal as possible in order to build solid bases for the study of keyboards.

Our first lemma states that applying a key can only affect a bounded part of the word around the cursor.

▶ **Lemma 24** (Locality). *Let $t = \sigma_1 \dots \sigma_n$ be a key. If $\langle u|v\rangle \xrightarrow{t} \langle u'|v'\rangle$, then $u[1, |u| - n]$ is a prefix of u' and $v[n + 1, |v|]$ is a suffix of v'.*

Furthermore, $u'[1, |u'| - n]$ is a prefix of u and $v'[n + 1, |v'|]$ is a suffix of v.

Then we formalize the fact that if the cursor is far enough from the extremities of the word, then we do not have edge effects.

▶ **Lemma 25** (Effectiveness far from the edges). *Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ a configuration and $\langle u_n|v_n\rangle = \langle u|v\rangle \cdot t$. If $n \leq \min(|u|, |v|)$, then $\langle u|v\rangle \xrightarrow{t}_e \langle u_n|v_n\rangle$, meaning that all the arrows and backspaces are applied effectively.*

The two next lemmas bound the variation in length of the configuration when applying a key.

▶ **Lemma 26** (Bounds on the lengths). *Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ a configuration and $\langle u_n|v_n\rangle = \langle u|v\rangle \cdot t$. Then*

$$|uv| - |t|_{\leftarrow} + \sum_{x\in A} |t|_x \quad \leq \quad |u_n v_n| \quad \leq \quad |uv| + \sum_{x\in A} |t|_x.$$

In particular $\big||u_n v_n| - |uv|\big| \leq n$. Moreover $\big||u_n| - |u|\big| \leq n$ and $\big||v_n| - |v|\big| \leq n$.

▶ **Lemma 27** (Length evolution without left edge effects). *Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ a configuration such that $|u| \geq n$. Let $\langle u_n|v_n\rangle = \langle u|v\rangle \cdot t$, then*

$$|u_n v_n| = |uv| - |t|_{\leftarrow} + \sum_{x\in A} |t|_x.$$

Then, we obtain the following lemma that can be used to show that some languages are not recognised by a keyboard.

▶ **Lemma 28.** *Let K be a keyboard with language L. Let $(\ell_n)_{n\in\mathbb{N}}$ be the sequence obtained by sorting the lengths of the words in L by increasing order. Then $(\ell_{n+1} - \ell_n)_{n\in\mathbb{N}}$ is bounded by $3\|K\|_\infty$.*

▶ **Example 29.** The languages $\left\{a^{n^2} \;\middle|\; n \in \mathbb{N}\right\}$ and $\{a^p \mid p \text{ prime}\}$ are not recognised by a keyboard.

The two following lemmas will be useful when studying effective executions.

▶ **Lemma 30.** *Let $t = \sigma_1 \dots \sigma_n$ be a key such that $\langle u|v\rangle \xrightarrow{t}_e \langle u_n|v_n\rangle$. Then, for all words x, y, $\langle xu|vy\rangle \xrightarrow{t}_e \langle xu_n|v_n y\rangle$.*

▶ **Lemma 31.** *Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ and $\langle x|y\rangle$ configurations such that $|u| = |x|$ and $|v| = |y|$. Then t acts efficiently from $\langle u|v\rangle$ if and only if it acts efficiently from $\langle x|y\rangle$.*

5 Key behaviour

This section aims at providing tools to describe the behaviour of a key. How can we formally express the intuitive fact that the i^{th} symbol of $c \cdot t$ was written by t or that the i^{th} symbol of c was moved by t? We are going to distinguish letters from t and c in order to keep track of where t writes its letters and how the letters of c were affected.

▶ **Definition 32** (Tracking function). *Let $\mathbb{Z}_t$ and $\mathbb{Z}_c$ be two duplicates of $\mathbb{Z}$. We denote by $\overline{k}$ the elements of $\mathbb{Z}_c$ and by $\widehat{k}$ the elements of $\mathbb{Z}_t$.*

We define the tracking functions, one for keys $f_t \colon S^* \to (S \cup \mathbb{Z}_t)^*$, defined as follows: $f_t(\sigma_1 \dots \sigma_n) = \sigma'_1 \dots \sigma'_n$ where

$$\sigma'_i = \begin{cases} \widehat{i} & \text{if } \sigma_i \in A \\ \sigma_i & \text{otherwise} \end{cases}$$

and one for configurations $f_c \colon \mathcal{C}(A) \to \mathbb{Z}_c^* \times \mathbb{Z}_c^*$ defined by

$$f_c(a_1 \dots a_k, b_1 \dots b_j) = \langle \overline{-k} \dots \overline{-1} | \overline{1} \dots \overline{j} \rangle.$$

By applying $f_t(t)$ to $f_c(c)$, we can keep track of which letters of the configuration and of the key were written, erased, or displaced, and where. We need two copies of $\mathbb{Z}$ to differentiate between the symbols of $f_t(t)$ (added by the key) and $f_c(c)$ (already in the configuration).

▶ **Definition 33.** *Let $\langle u|v\rangle$ be a configuration and t a key. We note $\langle u'|v'\rangle = \langle u|v\rangle \cdot t$ and $\langle x|y\rangle = f_c(u,v) \cdot f_t(t)$. We say that t writes its k^{th} symbol at position i from $\langle u|v\rangle$ if $\langle x|y\rangle_i = \widehat{k}$.*

▶ Remark 34. Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ a configuration and $1 \leq j < k \leq n$ integers. Then t writes its k^{th} symbol at position i from $\langle u|v\rangle$ if and only if $\sigma_{j+1} \dots \sigma_n$ writes its $(k-j)^{\text{th}}$ symbol at position i from $\langle u|v\rangle \cdot \sigma_1 \dots \sigma_j$. In particular, t writes its k^{th} symbol at position i from $\langle u|v\rangle$ if and only if $\sigma_k \dots \sigma_n$ writes its 1^{st} symbol from $\langle u|v\rangle \cdot \sigma_1 \dots \sigma_{k-1}$.

We defined an intuitive notion of writing the k^{th} symbol of t. In particular, if t writes its k^{th} symbol in i^{th} position from $\langle u|v\rangle$, then $\langle u'|v'\rangle_i = t_k$, as stated below.

▶ **Proposition 35.** *Let $t = \sigma_1 \dots \sigma_n$ be a key, $\langle u|v\rangle$ a configuration. We note*

$$\langle u_n|v_n\rangle = \langle u|v\rangle \cdot t \qquad \langle x'_n|y'_n\rangle = f_c(u,v) \cdot t$$
$$\langle u'_n|v'_n\rangle = \langle u|v\rangle \cdot f_t(t) \qquad \langle x_n|y_n\rangle = f_c(u,v) \cdot f_t(t)$$

Then $|u_n| = |x_n| = |u'_n| = |x'_n|$ and $|v_n| = |y_n| = |v'_n| = |y'_n|$. And for all $a \in A$,

$$\begin{aligned}
\langle u_n|v_n\rangle_j = a \text{ iff } & \langle x_n|y_n\rangle_j = \overline{k} \text{ and } \langle u|v\rangle_k = a && \text{or } \langle x_n|y_n\rangle_j = \widehat{k} \text{ and } t_k = a \\
\text{iff } & \langle u'_n|v'_n\rangle_j = a && \text{or } \langle u'_n|v'_n\rangle_j = \widehat{k} \text{ and } t_k = a \\
\text{iff } & \langle x'_n|y'_n\rangle_j = \overline{k} \text{ and } \langle u|v\rangle_k = a && \text{or } \langle x'_n|y'_n\rangle_j = a \\
& \text{(iff } a \text{ already in configuration} && \text{or } a \text{ added by } t\text{).}
\end{aligned}$$

Tracking functions are a convenient formalism to show some results on keyboards. Besides, they permit to take multiples points of view.

▶ **Corollary 36.** *Let t be a key and $\langle u|v\rangle$ a configuration. Then t writes its k^{th} symbol at position i from $\langle u|v\rangle$ if and only if $(\langle u|v\rangle \cdot f_t(t))_i = \widehat{k}$.*

▶ **Definition 37.** *Let t be a key, $\langle u|v\rangle$ a configuration. We say that t writes an a in i^{th} position from $\langle u|v\rangle$ if there exists k such that $t_k = a$ and t writes its k^{th} symbol in position i from $\langle u|v\rangle$. We say that t writes an a from $\langle u|v\rangle$ if t writes an a in some position from $\langle u|v\rangle$.*

Then, we obtain some results, which are direct consequences of Proposition 35.

▶ **Proposition 38.** *If t writes its k^{th} symbol in i^{th} position from $\langle u|v\rangle$ then $(\langle u|v\rangle \cdot t)_i = t_k$. In particular, if t writes an a in i^{th} position from $\langle u|v\rangle$ then $(\langle u|v\rangle \cdot t)_i = a$ and if t writes an a from $\langle u|v\rangle$ then $\langle u|v\rangle \cdot t$ contains an a.*

▶ **Proposition 39.** *Let t be a key and $\langle u|v\rangle$, $\langle u'|v'\rangle$ two configurations such that $|u| = |u'|$ and $|v| = |v'|$. Then t writes its k^{th} symbol in i^{th} position from $\langle u|v\rangle$ if and only if t writes its k^{th} symbol in i^{th} position from $\langle u'|v'\rangle$. In particular, t writes an a in j^{th} position from $\langle u|v\rangle$ if and only if t writes an a in j^{th} position from $\langle x|y\rangle$, and t writes an a from $\langle u|v\rangle$ if and only if t writes an a from $\langle x|y\rangle$.*

This proposition makes explicit the fact that keys cannot read the content of a configuration. This leads to the following characterization.

▶ **Proposition 40.** *Let t be a key, $a \in A$ and $\langle u|v\rangle$ a configuration containing no a. Let $\langle u'|v'\rangle = \langle u|v\rangle \cdot t$. t writes an a in position i from $\langle u|v\rangle$ if and only if $\langle u'|v'\rangle_i = a$. In particular, t writes an a from $\langle u|v\rangle$ if and only if $\langle u'|v'\rangle$ contains an a.*

Clearly, if the number of a's in a configuration increases after applying a key then this key writes an a.

▶ **Proposition 41.** *Let t be a key and $\langle u|v\rangle$ a configuration. If $|\langle u|v\rangle|_a < |\langle u|v\rangle \cdot t|_a$, then t writes an a from $\langle u|v\rangle$.*

Note that if a key behaves differently from two configurations, then there must be some edge effects. In what follows we focus on effective executions.

▶ **Proposition 42.** *Let t be a key and $\langle u|v\rangle$ and $\langle x|y\rangle$ two configurations on which t acts effectively. Then t writes its k^{th} symbol in i^{th} position from $\langle u|v\rangle$ if and only if t writes its k^{th} symbol in i^{th} position from $\langle x|y\rangle$. Therefore, t writes an a in position i from $\langle u|v\rangle$ if and only if t writes an a in position i from $\langle x|y\rangle$.*

In other words, a key always behaves the same way far from the edges of the configuration.

▶ **Definition 43.** *Let t be a key. We say that t ensures an a in position i far from the edges if there exists a configuration $\langle u|v\rangle$ such that t acts effectively on $\langle u|v\rangle$ and t writes an a in position i from $\langle u|v\rangle$.*

Then, we immediately obtain the following propositions.

▶ **Proposition 44.** *Let t be a key and $u, v \in A^*$ such that $|u| \geq |t|$ and $|v| > |t|$. Then t ensures an a far from the edges if and only if t writes an a from $\langle u|v\rangle$.*

▶ **Proposition 45.** *Let t be a key and $u, v \in A^*$ such that $|u| \geq |t|$ and $|v| \geq |t|$. If $|\langle u|v\rangle|_a < |\langle u|v\rangle \cdot t|_a$, then t ensures an a far from the edges.*

▶ **Proposition 46.** *Let t be a key which ensures an a far from the edges and $\langle u|v\rangle$ such that $|u| \geq |t|$ and $|v| \geq |t|$. Then $\langle u|v\rangle \cdot t$ contains an a.*

6 Characterisation of the classes

6.1 Languages of BEK (without the arrows)

To begin, we study keyboards that do not contain any arrows. As these keyboards has no ◀ operation, the right component of a configuration in an execution of $K \in \mathsf{BEK}$ (starting from $\langle\varepsilon|\varepsilon\rangle$) is always empty. Thus, in this part, we will sometimes denote u for the configuration $\langle u|\varepsilon\rangle$.

6.1.1 MK and EK

MK and EK are quite easy to understand. Indeed, since a key of a minimal keyboard K is just a word on A, $K \subset A^*$.

▶ Remark 47. Let $K = \{w_1, \ldots, w_n\}$ be a minimal keyboard. Then $\mathcal{L}(K) = (w_1 + \cdots + w_n)^+$.

EK languages are rather similar.

▶ **Lemma 48.** *Let $K = (T, F)$ be a* EK *keyboard. Then, $\mathcal{L}(K) = T^*F$ and this regular expression can be computed in $O(|K|)$.*

Thus, we can build in linear time a regular expression that recognises $\mathcal{L}(K)$.

6.1.2 BK and BEK

Some of the expressiveness of BEK comes from edge effects. For instance, finite languages are recognised by BK keyboards.

▶ **Example 49.** Let L be a finite language and $M = \max\{|w| \mid w \in L\}$. Then L is recognised by $K = \left\{\leftarrow^M w \;\middle|\; w \in L\right\}$.

These edge effects could make BEK languages quite complex. On the other hand, we show that BEK keys can be put in a particular form.

▶ **Lemma 50** (Normal form). *Let $t \in S^*$ be a key from* BEK*. Then there exist $m \in \mathbb{N}$ and $w \in A^*$ such that $t \sim \leftarrow^m w$. Further, m and w can be computed from t in polynomial time.*

Using this normal form, we understand that the action of a BEK key always consists in deleting a bounded number of letters at the end of the word, then adding a bounded number of letters. This reminds us of stacks. Following this intuition, we can easily encode the behaviour of a BEK keyboard into a pushdown automaton.

However, BEK is even more narrow since all languages of BEK are regular.

▶ **Theorem 51.** *Let K be a* BEK *keyboard. Then, $\mathcal{L}(K)$ is regular, and we can build an NFA $\mathcal{A}(K)$ recognising $\mathcal{L}(K)$ in polynomial time.*

▶ Remark 52. There are several ways to prove this result. One of them is to apply the pushdown automaton construction from the proof of Theorem 57 (presented later in the paper) in the particular case of BEK. The language of the BEK keyboard is then essentially the stack language of this automaton. A slight adaptation of the classical proof that the stack language of a pushdown automaton is rational then yields the result.

We choose to include another proof in this work, as it is elementary, not much longer than the one aforementioned, and seems more elegant to the authors.

We first show the result for languages of BK. Let K be an BK keyboard. The key idea is that at any point of the execution, we can split the word into:

- a prefix v of letters that will never be erased,
- a suffix x of letters which will eventually be erased.

As an example, in order to write abb with the keys $\leftarrow^2 abcc, \leftarrow^3 b$ and $\leftarrow^2 cbc$, we can write ab with t_1 and the remaining b with t_2. However, when applying t_1 and then t_2, t_2 erases too many letters and we obtain ab instead of abb. We therefore need a sequence of keys which does not erase the prefix ab, but replaces the two c by three letters, allowing us to apply t_2. This is done by applying t_3, replacing cc with cbc. We can therefore write abb with the sequence $t_1 t_3 t_2$.

Given a keyboard K of BK, we construct an automaton following this idea. Its states are integers from 0 to $\|K\|_\infty$, state i meaning that we have i extra letters to be erased at the end of the current word.

- A transition from i to j labelled by $w \in A^+$ means there exists a key erasing i letters and writing wx with $|x| = j$ (w is a part of the word we are trying to recognise and x is a suffix of extra letters to be erased).
- A transition from i to j labelled by ε means there exists a sequence of keys turning the suffix of extra letters of length i into one of length j, without erasing any of the other letters.

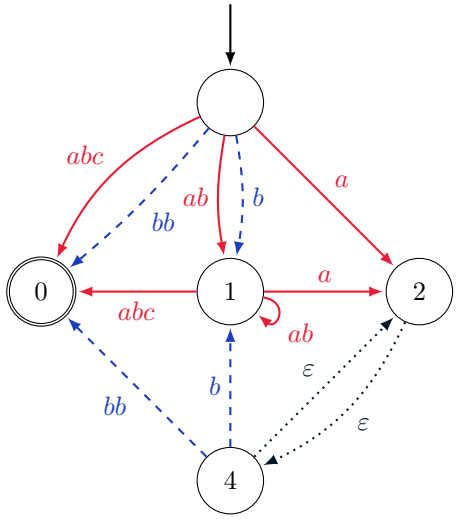

Figure 1 Automaton corresponding to keyboard $\{\leftarrow\texttt{abc}, \leftarrow^{4}\texttt{bb}\}$.

Figure 1 shows the automaton obtained from keyboard $\{\leftarrow\texttt{abc}, \leftarrow^{4}\texttt{bb}\}$ with this construction (slightly simplified as we erased some useless states).

The red transitions (full arrows) are the ones we can do with $\leftarrow\texttt{abc}$ and the blue (dashed) ones those we can do by applying $\leftarrow^{4}\texttt{bb}$. The black (dotted) transitions, are the ones we use to switch from i to j extra letters.

As an example, let us see how we can recognise *bababbb* with this automaton and this keyboard. We decompose it into $b \cdot a \cdot b \cdot ab \cdot abc$.

- We write a b with $\leftarrow^{4}\texttt{bb}$, leading us to state 1 (we need to erase the second b before to write the rest of the word).
- The key $\leftarrow\texttt{abc}$ then allows us to erase the second b and write a,leading to state 2 (we need to erase the bc suffix).
- A sequence of keys then allows us to switch from two extra letters (state 2 in the automaton) to four (state 4).
- The key $\leftarrow^{4}\texttt{bb}$ then allows us to erase four letters and write b, leaving us in state 1.
- The key $\leftarrow\texttt{abc}$ erases b and writes ab, leaving us in state 1.
- The key $\leftarrow\texttt{abc}$ erases the last letter and writes abc, with no extra letters, leaving us in state 0 in the automaton, where we can accept.

We need to construct the transitions of our NFA recognising the language of an BK keyboard. The construction of the transitions labelled by letters is rather straightforward, whereas the construction of the ε-transitions requires some arithmetic arguments, which are detailed in the full version of this paper.

We slightly adapt this construction to obtain an automaton for a keyboard of BEK.

- The state 0 is no longer final, and we add a final state Fin.
- For all states i and $t_f = \leftarrow^{i} w \in F$, we add a transition from i to Fin labelled by w. This simulates the action of, after producing vx (with $|x| = i$) with T, applying t_f.
- For all $t_f = \leftarrow^{r} w \in F$, we add a transition from Init to Fin labelled by w.

6.2 Languages of BLEK (without the right arrow)

In this section, we allow the use of ◀. With this symbol, we have the possibility to move into the word, and then erase or write letters. It opens a new complexity level.

Moreover, we provide a non-regular language of BLEK, hence showing that it is more expressive than BEK (see Theorem 51).

▶ **Example 53.** Let $K = \{aa◀, bb◀\}$. Then, $\mathcal{L}(K) = \{u\widetilde{u} \mid u \in (a+b)^+\}$, that is, K recognises the non-empty palindromes of even length.

Thus, we can represent context-free non-regular languages with BLEK (one can observe that the keyboard of Example 53 is actually even in LK).

However, a basic observation helps us to understand the behaviour of a key of BLEK: as we do not have the symbol ▶, we cannot go back to the right and all the letters to the right of the cursor are written forever. The following lemma can be proven easily by induction.

▶ **Lemma 54.** *Let $t = \sigma_1 \dots \sigma_n$ be a sequence of atomic operations, and $\langle u|v\rangle$ a configuration. Then, $\langle u|v\rangle \cdot t$ is of the form $\langle u'|v'v\rangle$.*

Then, we can make some assertions about a key observing its result over a configuration.

▶ **Lemma 55** (Independence from position)**.** *Let t be a key of* BLEK *and $\langle u|v\rangle$ a configuration. If t writes an a from $\langle u|v\rangle$, then for all configurations $\langle u'|v'\rangle$, t writes an a from $\langle u'|v'\rangle$.*

Proof. We set $t = \sigma_1 \cdots \sigma_n$. Let $\langle x|y\rangle$ be a configuration. By Proposition 39, we can assume it does not contain any a and by Remark 34, we can assume $\sigma_1 = a$ and t writes its first symbol from $\langle u|v\rangle$. We then define for all $1 \leq i \leq n$

$$\langle u_i|v_i\rangle = \langle u|v\rangle \cdot \sigma_1 \dots \sigma_i \qquad \text{and} \qquad \langle x_i|y_i\rangle = \langle x|y\rangle \cdot \sigma_1 \dots \sigma_i.$$

Let i be the smallest index such that u_i or x_i is empty, and $i = n$ if such an index does not exist (note that $i > 1$ as u_1 and x_1 contain the a written by σ_1). As t writes its first symbol from $\langle u|v\rangle$, $\sigma_1 \dots \sigma_i$ writes an a from $\langle u|v\rangle$ in some position j.

Further, $\sigma_1 \dots \sigma_i$ acts efficiently on $\langle u|v\rangle$ and on $\langle x|y\rangle$, thus by Proposition 42, t writes an a from $\langle x|y\rangle$ at position j. If $i = n$, the result is proven.

Otherwise, as either u_i or x_i is empty, we have $j > 0$ and $y_i[j] = v_i[j] = a$. By Lemma 54, y_n contains an a, while $\langle x|y\rangle$ does not by assumption, showing the result by Proposition 40. ◀

Moreover, we can refine Lemma 54.

▶ **Theorem 56** (BLEK fundamental)**.** *Let $t = \sigma_1 \dots \sigma_n$ be a sequence of atomic operations, and $\langle u|v\rangle$ a configuration. We set $\langle x_n|y_n\rangle = \langle \varepsilon|\varepsilon\rangle \cdot t$. Then $\langle u|v\rangle \cdot t$ is of the form $\langle u_n x_n|v_n v\rangle$ with y_n a subword of v_n and u_n a prefix of u.*

These observations help us to better understand BLEK. A key observation is that we can see the left part of the configuration as a stack, which can be modified, and the right part as the fixed one, just as the prefix of a word that has been read by an automaton. We can then recognise (the mirror of) a BLEK language with a pushdown automaton which guesses a sequence of keys, maintains the left part of the configuration in the stack and reads the right part of the configuration.

▶ **Theorem 57.** *Let K be a* BLEK *keyboard. Then, $\mathcal{L}(K)$ is context-free, and we can build a non-deterministic pushdown automaton $\mathcal{A}(K)$ recognising $\mathcal{L}(K)$ in polynomial time.*

The idea is to use the fact that what is written to the right of the cursor is unaffected by a key without right arrows (see Theorem 56).

We construct a pushdown automaton maintaining the invariant "After simulating the application of keys $t_1, \dots, t_n$ we have read a word v and have as stack content a word u such that $\langle \varepsilon | \varepsilon \rangle \cdot t_1 \cdots t_n = \langle u | \widetilde{v} \rangle$".

Atomic operations are easily translated to satisfy that invariant. Writing a letter comes down to pushing it on the stack, applying a backspace to deleting the stack head, and applying a left arrow to popping and reading the stack head.

After simulating a sequence of keys, the automaton pops and reads the content of its stack before to accept. At this point, the automaton has read the mirror of the word produced by this sequence of keys. We then simply construct a pushdown automaton recognising the mirror of the language of the previous one.

6.3 Languages of EAK (without backspace)

A third interesting class is EAK, where the backspace is not allowed. Thus, the size of a configuration does not decrease along an execution. The execution of such a keyboard can therefore be easily simulated on a linear bounded automaton, as stated in Theorem 60.

In all the proofs of this section we will say that t *writes an* a when t contains an a (as we have no $\leftarrow$ if t contains an a then this a will not be erased when applying t).

▶ **Lemma 58.** *Let $K = (T, F)$ be a* EAK *keyboard. Let $u, v \in A^*$, let $\tau \in (T \cup F)^*$ and let $\langle u' | v' \rangle = \langle u | v \rangle \cdot \tau$. Then uv is a subword of $u'v'$. In particular $|uv| \leq |u'v'|$.*

▶ **Lemma 59.** *Let $K = (T, F)$ be a* EAK *keyboard, let $w \in \mathcal{L}(K)$. There exists an execution $\tau = t_1 \cdots t_n \in T^*F$ such that $\langle \varepsilon | \varepsilon \rangle \cdot \tau = \langle u | v \rangle$ with $uv = w$ and $n \leq |w|^2 + 1$.*

Proof. Let $w \in \mathcal{L}(K)$, and let $u, v \in A^*$, $\tau = t_1 \cdots t_n \in T^*F$ be such that $w = uv$ and $\langle \varepsilon | \varepsilon \rangle \cdot \tau = \langle u | v \rangle$.

We show that if $n > |w|^2 + 1$ then there exists a shorter execution writing $\langle u | v \rangle$.

Suppose $n > |w|^2 + 1$, and for all $0 \leq i \leq n$ let $k_i = |\langle \varepsilon | \varepsilon \rangle \cdot t_1 \cdots t_i|$ and $\langle u_i | v_i \rangle = \langle \varepsilon | \varepsilon \rangle \cdot t_1 \cdots t_i$. By Lemma 58, the sequence (k_i) is nondecreasing. As $n > |w|^2 + 1$ and $k_n = |w|$, there exists $0 \leq i \leq n - |w|$ such that $k_i = k_{i+1} = \cdots = k_{i+|w|}$. Again by Lemma 58, we have $u_i v_i = u_{i+1} v_{i+1} = \cdots = u_{i+|w|} v_{i+|w|}$. If $u_i v_i = w$ then the execution $t_1 \cdots t_i$ writes w.

If $u_i v_i \neq w$ then $k_i < |w|$. There are $|w| > k_i$ configurations $\langle u' | v' \rangle$ such that $u'v' = u_i v_i$, thus there exist $i \leq j_1 < j_2 \leq i + |w|$ such that $\langle u_{j_1} | v_{j_1} \rangle = \langle u_{j_2} | v_{j_2} \rangle$.

As a result, the execution $t_1 \cdots t_{j_1} t_{j_2+1} \dots t_n$ writes w.

The lemma is proven. ◀

▶ **Theorem 60.** *For all keyboards $K = (T, F)$ of* EAK *we can construct a linear bounded automaton $\mathcal{A}(K)$ of polynomial size recognising $\mathcal{L}(K)$.*

Proof. We construct $\mathcal{A}(K)$ the linear bounded automaton which, given an input w, proceeds as follows:

- It divides the tape in three parts: one to memorize the input (of linear size), one to simulate an execution of K (of linear size as well by Lemma 58) and one containing a counter (of logarithmic size).
- It guesses a sequence of keys of T followed by a key of F and computes their effect on the fly. After the application of each key, the counter is incremented.

- If the counter goes beyond $|w|^2 + 1$, then the automaton rejects.
- If not, the automaton compares the obtained word to w, accepts if they are equal, and rejects otherwise.

This machine guesses a sequence of at most $|w|^2 + 1$ keys and accepts if the word obtained by their actions is the input.

Clearly if a word is accepted by $\mathcal{A}(K)$ then it is in $\mathcal{L}(K)$. Conversely, if a word w is in $\mathcal{L}(K)$ then by Lemma 59 there exists an execution of length at most $|w|^2 + 1$ accepting it, thus $\mathcal{A}(K)$ can guess this execution and accept w.

As a result, $\mathcal{L}(K) = \mathcal{L}(\mathcal{A}(K))$. ◀

7 Comparison of the keyboard classes

The characterisations that we provide give us some information about each class independently. We now compare the subclasses of BEAK in order to find out which inclusions hold between them. One of these inclusions, between BAK and BEAK, is especially interesting since it shows that keyboards with entry are strictly more powerful than automatic ones.

We decompose our results into the following propositions. Those establish that a class is included in another if and only if that same inclusion holds between their sets of operators, except possibly for the inclusion of EK and BEK in BAK, which we do not prove or disprove.

To start with, we show that a class containing the left arrow cannot be included in a class lacking it. This is a direct consequence of Example 53 and Theorem 51 as we have a language of LK which is not rational, and thus not in BEK.

▶ **Proposition 61.** LK $\not\subseteq$ BEK.

We continue with the two next propositions, showing that a class containing the entry cannot be included in a class excluding it, except possibly for BAK.

▶ **Proposition 62** (EK $\not\subseteq$ BLK)**.** EK *is not included in* BLK.

▶ **Proposition 63** (EK $\not\subseteq$ AK)**.** EK *is not included in* AK.

Then we prove that a class with $\leftarrow$ cannot be included in a class without $\leftarrow$.

▶ **Proposition 64** (BK $\not\subseteq$ EAK)**.** BK *is not included in* EAK.

The next proposition states that a class containing ▶ cannot be included in a class lacking it.

▶ **Proposition 65** (AK $\not\subseteq$ BLEK)**.** AK *is not included in* BLEK.

And finally we show that, except possibly for EK and BEK, a class with entry cannot be included in a class without entry.

▶ **Proposition 66** (LEK $\not\subseteq$ BAK)**.** LEK *is not included in* BAK.

The inclusion Hasse diagram of all subclasses of BEAK and traditional language classes is displayed in Figure 2.

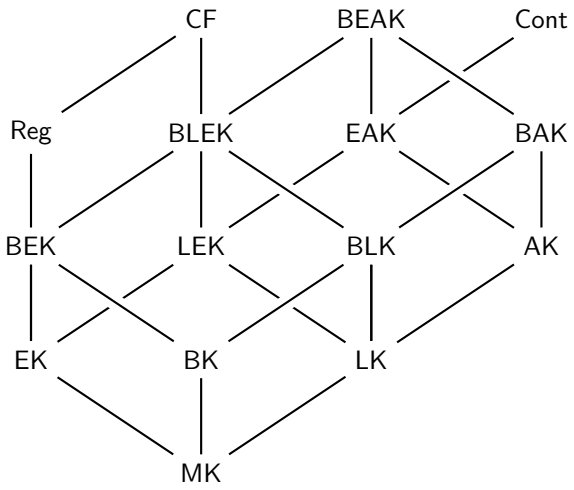

Figure 2 Hierarchy of language classes.

8 Complexity results

In this section we establish some complexity upper bounds on the membership and universality problems for various keyboard classes. The following three propositions are direct consequences of the known complexity bounds of the models which we translated keyboards into (in Remark 47, Lemma 48, Theorem 51 and Theorem 57).

▶ **Proposition 67.** *The membership problem on* MK *and* EK *is in PTIME. The universality problem is in PTIME on* MK *and PSPACE on* EK*.*

The problem for EK seems simple: it amounts to deciding, given two finite sets of words T and F, if T^*F is universal.

▶ **Proposition 68.** *The membership problem over* BEK *is in PTIME, and the universality problem over* BEK *in PSPACE.*

▶ **Proposition 69.** *The membership problem over* BLEK *is in PTIME.*

For BK keyboards, we prove that there exists a word not accepted by a given BK keyboard if and only if there exists one of polynomial length.

▶ **Proposition 70.** *The universality problem for* BK *keyboards is in* coNP*.*

For EAK keyboards, we know by Lemma 59 that every word w recognised by a EAK keyboard can be written with an execution of length polynomial in $|w|$, hence this proposition:

▶ **Proposition 71.** *The membership problem for* EAK *keyboards is in NP.*

9 Closure properties

In this section, we study closure properties of keyboard classes. We selected three operators, the union and the intersection, as they are the most natural closure operators, and the mirror, under which some classes are stable.

▶ **Proposition 72** (Mirror)**.** MK*,* AK *and* EAK *are stable by mirror.* EK*,* BK*,* BEK *and* BLK *are not stable by mirror.*

▶ **Proposition 73** (Intersection). *None of the keyboard language classes are stable by intersection.*

▶ **Proposition 74** (Union). *None of the keyboard language classes are stable by union.*

We end this section with an undecidability result, showing that intersecting keyboards can lead to highly complex languages. This shows another link with context-free languages, as the emptiness of the intersection of two context-free languages is undecidable as well.

▶ **Proposition 75** (Intersection emptiness problem). *The following problem is undecidable:*
Input: *K_1, K_2 two* LK *keyboards.*
Output: *Is $\mathcal{L}(K_1) \cap \mathcal{L}(K_2)$ empty?*

10 Conclusion

A natural question when it comes to models of computation is what we can do without any memory or any information on the current state of the system. We initiated a line of research aiming at studying such "blind" models. The one we considered here, keyboards, proved to be mathematically complex and interestingly orthogonal in expressiveness to several of the most classical models. We have established a number of properties of keyboards, as well as a vocabulary facilitating their study. We separated almost all classes and compared their expressiveness, thereby uncovering the lattice of their power.

Future work

As keyboards are a completely new model, there are many open problems we are working on or intend to address. We conjecture that EK is not included in BAK, but we do not have a proof. We also conjecture that not all AK languages are algebraic (the language generated by $\{a\blacktriangleright\blacktriangleright, b\blacktriangleleft\blacktriangleleft\}$ is a candidate as a counter-example) and that BEAK does not contain all rational languages ($a^* + b^*$ being a potential counter-example). We plan on extending the set of operations to add, for instance, a right erasing operator, symmetric to $\leftarrow$. It could also be interesting to study the semantics in which we forbid non-effective operations. Finally, we could equip the model with states and transitions labelled with keys. We would then have more control over which keys are applied at which times, thus increasing the expressiveness of the model and facilitating its study.

References

1 L. Babai and E. Szemeredi. On the complexity of matrix group problems i. In *25th Annual Symposium on Foundations of Computer Science, 1984.*, pages 229–240, 1984. doi:10.1109/SFCS.1984.715919.

2 Martin Beaudry. Membership testing in commutative transformation semigroups. *Information and Computation*, 79(1):84–93, 1988. doi:10.1016/0890-5401(88)90018-1.

3 Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*, volume 129 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2010.

4 Petr Jančar, Frantiek Mráz, and Martin Plátek. Forgetting automata and the Chomsky hierarchy. In *Proc. SOFSEM'92*, 1993.

5 Petr Jančar, František Mráz, and Martin Plátek. Characterization of context-free languages by erasing automata. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992*, pages 307–314, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

6 Rajeev Motwani John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Pearson Education, Limited, 2013.

7 Neil D. Jones and William T. Laaser. Complete problems for deterministic polynomial time. *Theoretical Computer Science*, 3(1):105–117, 1976. doi:10.1016/0304-3975(76)90068-2.

8 Michael S Paterson. Unsolvability in 3× 3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

9 Burchard von Braunmühl and Rutger Verbeek. Finite-change automata. In K. Weihrauch, editor, *Theoretical Computer Science 4th GI Conference*, pages 91–100, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

A Comparisons between classes

▶ **Proposition 62** (EK $\not\subseteq$ BLK)**.** EK *is not included in* BLK*.*

Proof. The language $L = (a^2)^*(b + b^2)$ is clearly in EK. We first show that a keyboard K of BLK recognising L does not use the ◀ operation. The idea is that if we could reach a configuration with letters to the right of the cursor, we could then apply a sequence of keys writing b^2 from $\langle\varepsilon|\varepsilon\rangle$, which would yield a word with letters after the second b (thus not in L) by Theorem 56.

K is therefore an BK keyboard which can write b^2, hence there exists a sequence of keys of K whose normal form is $\leftarrow^k b^2$ for some k. We reach a contradiction by considering a word $w \in L$, applying τ, and distinguishing cases according to the parity of k. ◀

▶ **Proposition 63** (EK $\not\subseteq$ AK)**.** EK *is not included in* AK*.*

Proof. Consider the language $\{a\}$. It is recognised by the keyboard $(\emptyset, \{a\})$ of EK.

Suppose there exists K a keyboard of AK recognising $\{a\}$. Then there exists $t \in K$ exactly containing an a. As K does not contain any $\leftarrow$, applying t twice from $\langle\varepsilon|\varepsilon\rangle$ yields a configuration with two a, and thus a word outside of $\{a\}$. We obtain a contradiction. ◀

Proof of Proposition 64

Define $t_a = \leftarrow a\Diamond\blacklozenge$ and $t_b = \leftarrow\leftarrow b\Diamond\blacklozenge\blacklozenge$, and $L_{\Diamond\blacklozenge}$ as the language recognised by the keyboard $\{t_a, t_b\}$. $L_{\Diamond\blacklozenge}$ is in BK, and we show that it is not in EAK using mainly the non-erasing property of EAK keyboards.

In all that follows, we will use the notations $t_a = \leftarrow a\Diamond\blacklozenge$ and $t_b = \leftarrow\leftarrow b\Diamond\blacklozenge\blacklozenge$.

We define $L_{\Diamond\blacklozenge}$ as the language recognised by the BK keyboard $\{t_a, t_b\}$.

▶ **Lemma 76.** *Let $x = x_1 \cdots x_n \in \{a, b\}^+$, we have*

$$\langle\varepsilon|\varepsilon\rangle \cdot t_{x_1} \cdots t_{x_n} = x_1 w_{x_1x_2} x_2 w_{x_2x_3} x_3 \cdots w_{x_{n-1}x_n} x_n v_{x_n}$$

with $w_{aa} = w_{bb} = \Diamond$, $w_{ab} = \varepsilon$, $w_{ba} = \Diamond\blacklozenge$, $v_a = \Diamond\blacklozenge$ and $v_b = \Diamond\blacklozenge\blacklozenge$.

In particular, for all $u_1, u_2, u_3 \in A^*$, if $u_1 b u_2 a u_3 \in L_{\Diamond\blacklozenge}$ then u_2 contains a $\blacklozenge$.

▶ **Proposition 64** (BK $\not\subseteq$ EAK)**.** BK *is not included in* EAK*.*

Proof. Suppose there exists a keyboard $K = (T, F)$ of EAK recognising $L = L_{\Diamond\blacklozenge}$. In this proof, we will use the letter k to denote keys of K, in order to avoid confusion with the keys t_a and t_b.

As $a\Diamond\blacklozenge \in L$, there exists $\tau_f \in T^*F$ yielding that word.

Moreover, $w = (a\Diamond)^{3\|K\|_\infty} a\Diamond\blacklozenge \in L$ (obtained by applying t_a $3\|K\|_\infty$ times). Hence, there exists an execution of K writing that word. By Lemma 58, as $|k|_a \leq \|K\|_\infty$ for all $k \in T \cup F$, this execution contains at least three keys containing an a, including at least two in T (as the execution only has one final key).

As w contains a single $\blacklozenge$, there exists a key k_a of T writing an a, but writing neither $\blacklozenge$ nor b. By applying k_a then τ_f, we obtain a word of L containing a single $\blacklozenge$ and no b. By lemma Lemma 76 this word must be of the form $(a\Diamond)^i a\Diamond\blacklozenge$. We infer that k_a contains as many a and $\Diamond$. Let n be its number of a.

We can similarly show, using the word $(b\Diamond)^{3\|K\|_\infty} b\Diamond\blacklozenge\blacklozenge$, that there exists $k_b \in T$ containing as many b and $\Diamond$ but neither $\blacklozenge$ nor a. Let m be its number of b (and of $\Diamond$).

Let $\tau = k_a k_b$. We write

$$\langle u|v\rangle = \langle \varepsilon|\varepsilon\rangle \cdot \tau \quad \text{and} \quad \langle u'|v'\rangle = \langle u|v\rangle \cdot \tau_f$$

We have that $u'v'$ contains a single $\blacklozenge$. As $u'v' \in L$, $u'v'$ has to be of the form $wa\Diamond\blacklozenge$ with w not containing any $\blacklozenge$, $|w|_a = n$, $|w|_b = m$ and $|w|_\Diamond = n + m$.

As wa contains at least a b, wa it is of the form $u_1 b u_2 a$, thus by Lemma 76, u_2 contains a $\blacklozenge$, yielding a contradiction. As a conclusion, $L \notin \mathsf{EAK}$. ◀

Proof of Proposition 65

Consider the keyboard $K = \{\mathtt{a}\blacktriangleleft^2\blacktriangleright\mathtt{b}\}$.

By applying $a\blacktriangleleft^2\blacktriangleright b$ on $\langle\varepsilon|\varepsilon\rangle$ we obtain $\langle ab|\varepsilon\rangle$, and by applying it on a configuration of the form $\langle ub|v\rangle$ we obtain $\langle ubb|av\rangle$. Hence, after applying it n times on $\langle\varepsilon|\varepsilon\rangle$ we get $ab^{n+1}a^n$. The language of K is therefore $L = \left\{ab^{n+1}a^n \;\middle|\; n \in \mathbb{N}\right\}$.

▶ **Lemma 77.** *Let K be a keyboard of* BLEK*. If K recognises L then, for all $\tau \in T^*$, $\langle\varepsilon|\varepsilon\rangle \cdot \tau$ is of the form $\langle u|a^k\rangle$.*

Proof. As $abba \in L$, there exists $\tau \in T^*$ and $t_f \in F$ such that $\langle\varepsilon|\varepsilon\rangle \cdot \tau t_f = \langle x|y\rangle$ with $xy = abba$.

Let $\langle u|v\rangle$ be a configuration reachable by a sequence of keys of τ with v containing a b. By Theorem 56, $\langle u|v\rangle \cdot \tau t_f$ is of the form $\langle u'x|v'v\rangle$ with y a subword of v'. As a consequence, $abba$ is a subword of $u'xv'$. With the assumption, we get that $abbab$ is a subword of $u'xv'v$, which contradicts the fact that $u'xv'v \in L$. ◀

▶ **Proposition 65** (AK $\not\subseteq$ BLEK)**.** AK *is not included in* BLEK*.*

Proof. Let $K = (T, F)$ be a keyboard of BLEK, suppose it recognises L. Let $\tau \in T^*$ and $t_f \in F$. We set:

$$\langle x|y\rangle = \langle\varepsilon|\varepsilon\rangle \cdot t_f \quad \text{and} \quad \langle u|v\rangle = \langle\varepsilon|\varepsilon\rangle \cdot \tau.$$

There exists $n \in \mathbb{N}$ such that $xy = ab^{n+1}a^n$. By Theorem 56, $\langle\varepsilon|\varepsilon\rangle \cdot \tau t_f = \langle u|v\rangle \cdot t_f$ is of the form $\langle u'x|v'v\rangle$ with y a subword of v'.

As $ab^{n+1}a^n$ is a subword of xv' and $u'xv'v \in L$, u' is necessarily empty (otherwise u' would contain an a and as ab is a subword of xv', aab would be a subword of $u'xv'v$, contradicting $u'xv'v \in L$).

By Lemma 26, $|v'v| - |v| \leq |t| \leq \|K\|_\infty$, i.e., $|v'| \leq \|K\|_\infty$. Furthermore, again by Lemma 26, $|x| \leq \|K\|_\infty$.

By Lemma 77, v is of the form a^k. Hence, all b in $u'xv'v$ are in xv', thus $u'xv'v$ contains at most $2\|K\|_\infty$ b.

We have shown that all words in $\mathcal{L}(K)$ contain at most $2\|K\|_\infty$ b, contradicting $\mathcal{L}(K) = L$ as the number of b of words of L is unbounded. ◀

Proof of Proposition 66

Consider the following language over $A = \{a, b, c\}$:

$L = L_1 \cup L_2$ with $L_1 = \left\{wc\widetilde{w} \mid w \in \{a, b\}^*\right\}$ and $L_2 = \left\{wcc\widetilde{w} \mid w \in \{a, b\}^*\right\}$.

We are going to prove that L is in LEK but not in BAK. L is recognised by $K = (\{\texttt{aa}\blacktriangleleft, \texttt{bb}\blacktriangleleft\}, \{\texttt{c}, \texttt{cc}\})$, then L is in LEK.

We now show that $L \notin$ BAK. The intuition is as follows:

Suppose we have a keyboard K of BAK recognising L. As we want to recognise palindromes, we need to stay close to the centre in order to always modify both halves of the word.

If the word is large enough, this means that we have to get far from the edges. In particular, there is a key t writing an a far from the edges (even two a, as we have to stay in the language).

We study the behaviour of this key on b^ncb^n and b^nccb^n with large n. The cursor being far from the edges, t behaves the same way in both cases. In particular, we make the following remarks:

- The maximal distance d between two a will be the same in both words. Thus, the resulting words have either both two c or both one c in the centre.
- In both cases the key adds a number of letters δ, and thus if b^ncb^n is turned into a word of even length, then b^nccb^n is turned into a word of odd length, and vice-versa.
  As a result, they have different numbers of c in the centre.

As those facts are contradictory, we conclude that there cannot exist such a keyboard.

▶ **Proposition 66** (LEK $\not\subseteq$ BAK)**.** LEK *is not included in* BAK*.*

B Complexity

Proof of Proposition 70

▶ **Lemma 78.** *Let K be an* BK *keyboard. For all $\tau \in K^*$ there exists $\tau' \in K^*$ whose normal form is $\leftarrow^{k'} w'$ with $k' \leq \|K\|_\infty$ such that $\varepsilon \cdot \tau = \varepsilon \cdot \tau'$.*

Proof. We proceed by induction on τ. The result hold for $\tau = \varepsilon$ (by taking $\tau' = \varepsilon$). Let $\tau \in K^*$, suppose there exists $\tau' \in K^*$ whose normal form is $\leftarrow^{k'} w'$ with $k' \leq \|K\|_\infty$ such that $\varepsilon \cdot \tau = \varepsilon \cdot \tau'$. Let $t \in K$, we study the sequence $t\tau$. We need to provide τ'' such that $\varepsilon \cdot t\tau = \varepsilon \cdot \tau''$ and with normal form $\leftarrow^{k''} w''$ with $k'' \leq \|K\|_\infty$.

There exist k and w such that $\leftarrow^k w$ is the normal form of t. Let $n = |w| - k'$.

- If $n > 0$, then $t\tau'$ is equivalent to $\leftarrow^k w[1, n]w'$ and we can set $\tau'' = t\tau'$ (as $n > 0$, we have $k' < |w| < \|K\|_\infty$).
- Otherwise, $\varepsilon \cdot t\tau = w \cdot \tau = w \cdot \tau' = w' = \varepsilon \cdot \tau$ and we can set $\tau'' = \tau'$.

The result is proven. ◀

▶ **Proposition 70.** *The universality problem for* BK *keyboards is in* coNP*.*

Proof. We prove that if a BK keyboard K is not universal, then there is a word of length at most $\|K\|_\infty + 1$ it does not recognise.

Let K be a keyboard of BK, suppose there exists $w \in A^*$ not recognised by K. We take w of minimal length. If $|w| \leq \|K\|_\infty + 1$ then the property holds.

If $|w| > \|K\|_\infty + 1$ then there exist $a \in A, v \in A^+$ such that $av = w$. As we assumed w to be of minimal length, v is recognised by K. By Lemma 78, there exist $k \leq \|K\|_\infty$, $\tau \in K^*$ whose normal form is $\leftarrow^k v$, such that $\varepsilon \cdot \tau = v$.

As $\left|a^{k+1}\right| = k+1 \leq \|K\|_\infty + 1 < |w|$, a^{k+1} is recognised by K. Let τ' be such that $\varepsilon \cdot \tau' = a^{k+1}$, then we have $\varepsilon \xrightarrow{\tau'} a^{k+1} \xrightarrow{\tau} av = w$.

This contradicts the fact that w is not recognised by K. The property is proven. ◀

C Closure properties

Proof of Proposition 72

▶ **Proposition 72** (Mirror). MK, AK *and* EAK *are stable by mirror.* EK, BK, BEK *and* BLK *are not stable by mirror.*

For MK the result is clear as the mirror of $\mathcal{L}(K)$ is $\mathcal{L}\left(\widetilde{K}\right)$ when K is an MK keyboard.

As for AK and EAK, we first define the mirror of an atomic operation: the mirror of ◀ is ▶ (and vice-versa) and the mirror of a is $a◀$. By turning every atomic operation in a keyboard into its mirror, we obtain a keyboard recognising the mirror language.

The language $L = b^*a$ is in EK and BK but its mirror is not in BEK, thus EK, BK and BEK are not stable under mirror.

The language $L = (b + b^2)a^*$ is recognised by the BLK keyboard

$$K = \left\{\leftarrow^2 \mathtt{a}◀\mathtt{b}, \leftarrow^2 \mathtt{a}◀\mathtt{bb}, \leftarrow^2 \mathtt{b}, \leftarrow^2 \mathtt{bb}\right\}$$

but its mirror $a^*(b + b^2)$ is not in BLK as is shown in the proof of Proposition 62.

Proof of Proposition 73

▶ **Proposition 73** (Intersection). *None of the keyboard language classes are stable by intersection.*

For MK and EK, we use the intersection of $(ab + ba + bb)^*$ and $(ba + b)^*$ as our counterexample.

For BK and BEK, we use again the language $L_{\Diamond\blacklozenge}$ from the proof of Proposition 64. We also define $L' = (a + b + \Diamond)^+\blacklozenge^2$ recognised by the BK keyboard

$$\left\{\leftarrow^2 a\blacklozenge^2, \leftarrow^2 b\blacklozenge^2, \leftarrow^2 \Diamond\blacklozenge^2\right\}.$$

Thus L and L' are in BK, and we show that their intersection is not.

For the other classes, we show that $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ is not in BEAK. Further, it is the intersection of the following LK languages:

- $(a + b)^* c^*$, recognised by $\{a, b, c◀\}$,
- $a^*(b + c)^*$, recognised by $\{a, b◀, c◀\}$,
- $\{w \in (a + b + c)^* \mid |w|_a = |w|_b\}$, recognised by $\{ab, ba, c, ◀\}$,
- $\{w \in (a + b + c)^* \mid |w|_b = |w|_c\}$, recognised by $\{bc, cb, a, ◀\}$.

Proof of Proposition 74

▶ **Proposition 74** (Union). *None of the keyboard language classes are stable by union.*

To start with, we consider the languages a^* and b^*, both in MK, and prove that their union is neither in BLEK nor in EAK.

▶ **Lemma 79.** *The language $L = a^* + b^*$ is not in* BLEK.

Proof. Suppose there exists a keyboard $K = (T, F)$ of BLEK recognising L. Then there exists $\tau_a \in T^*$, $f_a \in F$ such that $\langle \varepsilon|\varepsilon\rangle \cdot \tau_a f_a = \langle u_a|v_a\rangle$ with $u_a v_a = a$.

There also exists $\tau_b \in T^*$, $f_b \in F$ such that $\langle \varepsilon|\varepsilon\rangle \cdot \tau_b f_b = \langle u_b|v_b\rangle$ with $u_b v_b = b^{1+\|K\|_\infty(|\tau_a|+2)}$.

By Lemma 24, applying τ_b to $\langle \varepsilon|\varepsilon\rangle$ yields a configuration with at least $1 + \|K\|_\infty(|\tau_a| + 1)$ b. We apply $\tau_b\tau_a f_a$ to $\langle \varepsilon|\varepsilon\rangle$, by Theorem 56 the resulting configuration contains an a, and as $\tau_a f_a$ can only erase at most $(|\tau_a| + 1)\|K\|_\infty$ letters, it contains a b. This is impossible, as the resulting word should be in L. ◀

▶ **Lemma 80.** *The language $L = a^* + b^*$ is not in* EAK.

Proof. Suppose there exists $K = (T, F)$ a keyboard of EAK recognising L. As $a^{\|K\|_\infty+1}$ and $b^{\|K\|_\infty+1}$ are both in L, there exist $t_a, t_b \in T$ such that t_a writes an a and t_b a b (and those letters are never erased, as we do not have $\leftarrow$). Let $f \in F$, $t_a t_b f$ writes a word containing both a and b, thus not in L. ◀

To finish, we define the language $L = L_a \cup L_b$ with $L_a = \{a^n c a^n \mid n \in \mathbb{N}\}$ and $L_b = \{b^n c b^n \mid n \in \mathbb{N}\}$. We can prove that L is not in BEAK, showing that all BAK and BEAK are not stable by intersection.

Proof of Proposition 75

▶ **Proposition 75** (Intersection emptiness problem)**.** *The following problem is undecidable:*
Input: *K_1, K_2 two* LK *keyboards.*
Output: *Is $\mathcal{L}(K_1) \cap \mathcal{L}(K_2)$ empty?*

Proof. We reduce the Post Correspondence Problem. Let $(u_i, v_i)_{i\in[\![1,n]\!]}$ be a PCP instance. For all $i \in [\![1, n]\!]$, let $u_i^{\blacklozenge}$ and $v_i^{\blacklozenge}$ be u_i and v_i where we added a $\blacklozenge$ at the right of every letter, *i.e.*, if $u_i = a_1 a_2 \cdots a_n$ then $u_i^{\blacklozenge} = a_1\blacklozenge a_2\blacklozenge \cdots a_n\blacklozenge$.

We set for all $i \in [\![1, n]\!]$, $t_i = u_i^{\blacklozenge}\widetilde{v_i^{\blacklozenge}}\blacktriangleleft^{2|v_i|}$. Let

$$K_{pal} = \{aa\blacktriangleleft \mid a \in A \cup \{\blacklozenge\}\} \cup \{\varepsilon\}$$

K_{pal} recognises the language of even palindromes over $A \cup \{\blacklozenge\}$. Now let

$$K = \{t_i \mid i \in [\![1, n]\!]\}.$$

We easily show that $\mathcal{L}(K) \cap \mathcal{L}(K_{pal}) \neq \emptyset$ if and only if $(u_i, v_i)_{i\in[\![1,n]\!]} \in$ PCP. ◀