Communicating Finite Automata System and Tally Languages

M2 - Internship report

Bruno Guillon[a],
internship supervised by Christian Choffrut[b]

[a] *Université Nice-Sophia Antipolis and École Normale Supérieure de Lyon, France*
[b] *L.I.A.F.A (Laboratoire d'Informatique Algorithmique, Fondements et Applications), Université Paris VII, 2 pl. Jussieu, 75251 Paris, France*

Acknoledgements: Many thanks to Christian Choffrut to having directed my work during this internship. Many thanks also to Noëlle Delgado and Brigitte Bloise for bureaucratic work. Thanks to Timo Jolivet for having encouraged me to participate to *EJCIM 2012* and simplified my registration. Many thanks to all persons I encountered in *LIAFA* during my stage.

Contents

1 Introduction **2**

2 Definitions, notations and first results **4**
- 2.1 Generalities 4
  - 2.1.1 Words and languages 4
  - 2.1.2 Finite automata 4
  - 2.1.3 Known results 5
- 2.2 Finite Automata System 6
  - 2.2.1 General definitions 6
  - 2.2.2 One-way Unary Finite Automata System 7
- 2.3 Tri-phase Sweeping Unary Finite Automata System 7
  - 2.3.1 Tri-phase Sweeping Unary Deterministic Finite Automata 8
  - 2.3.2 Tri-phase Sweeping Unary Deterministic Finite Automata Systems 10

3 Main Result **11**
- 3.1 $TSUDFAS$ with a constant number of communications 11
- 3.2 $2DFAS$ simulation by $TSUDFAS$ 14

4 Conclusion **16**

Bibliography **17**

1. Introduction

Finite Automaton (FA) is one of the simplest computing model in computer science. For a long time, researcher have studied it and its derivated forms: deterministic/nondeterministic, one-way/two-way, pushdown, self-verifying, multihead, probabilistic... However, despite the simplicity of the model, a lot of relevant questions remain open.

During the 6-months intership I did in LIAFA, under direction of professor C. Choffrut, I worked on *Communicating Finite Automata System* (FAS for short), which is a parallel improvement of the classical FA. (In particular, FASs accept at least the class of regular languages.) This computing model was introduced by T. Jurdzinski at the end of the XX-th century [32]. FASs are synchronized[1] parallel automata working on a shared read-only input, which are allowed to communicate each other, thanks to their transition functions. As for the classical FA, one may consider deterministic/nondeterministic or one-way/two-way versions of the model. Another point of view is to see FASs as *multi-head automata*, but with separate finite controls and transition functions for each head, then communications are not done by sharing state information rather by sending (broadcasting) messages. The goal of having introduced such a difference to multi-head automaton, is to save a control on the the number of communications in order to limit it: clearly, if this communication number is unbounded, then FASs are equivalent to multi-head FAs. It is well known that in parallel computing, the cost of communications is much more relevant than that of the local computing. Therefore parallel devices with bounded communication raise interesting questions in Complexity Theory: what problems can solve a FAS if it mays exchange at most a fixed number of messages?

T. Jurdzinski proved that there exists a gap between $\mathcal{O}(1)$ and $\mathcal{O}(\log n)$ messages (*resp.* $\mathcal{O}(\log\log\log n)$) where n is the size of the input, for the one-way (*resp.* two-way) FAS [32, 33]. He also exhibates an example of one-way deterministic FAS, with only two automata and with communication bounded by 1 (*i.e.* at most one message is sent in each computation) that accepts a non-regular language. However its witness language (which is in fact the well known $a^n b^n$ language) is define over two-letters alphabet. In the litteratur we may find several interesting differences between result on unary (also called $Tally$) languages (*i.e.* languages defined over a one-letter alphabet) and that on languages defined over bigger alphabet. The probably most famous example of such a difference is the collapse between *regular* and *context-free* language classes in unary case [20]. Starting with these observations in mind, one may raise the question of what may accept unary FASs with bounded communications. I started the internship with the following quadruple conjecture:

Conjecture 1. *If a $Tally$ language $\mathcal{L}$ is accepted by a one-way/two-way deterministic/nondeterministic FAS with communication bounded by a constant, then $\mathcal{L}$ is regular.*

In fact, the one-way case turn out to be already solved by M. Harrisson & O. Ibarra [24]. They proved a more general result: one-way nondeterministic multi-head automata over unary alphabet accept exactly the class of regular languages. Hence in that case (one-way), the number of communications exchanged by the automata surprisely does not affect the power of the computing model. In contrast, for the two-way case, we proved that $\log(n)$ communications suffies to deterministically accept a non-regular $Tally$ language (see Section **??**). The problem turn out to be more difficult than expected. We find a positive answer to the Conjecture 1 for the case of two-way deterministic FAS. Nevertheless,

[1] T. Jurdzinski studied non-synchronized systems too [34], but these models turn out to be more complicated to describe and not so interesting. In this report we only consider synchronized systems.

the problem for the two-way nondeterministic case remains open.

I start by giving some formal definitions and basic remarks in automata theory in Section 2, introducing also a new variant of the model, which will be usefull for our proof. Then I will prove our main result (positive answer to Conjecture 1 for the two-way deterministic case) in two time in Section 3. Finally I will give others remarks and research experience I had during this internship, raising some other questions in relation with the topic, in Section 4.

2. Definitions, notations and first results

2.1. Generalities

2.1.1. Words and languages

We suppose the reader is familiar with Language Theory, in particular we do not do any recalls on definitions of *context-free* or *regular* languages classes. Our notations are usual: ϵ is the empty word; $|u|$ is the length of word u; $u[i]$ is the i-th letter of word u; uv is the concatenation of words u and v; u^i is the i-th iterate of self concatenetion of word u ($u^0 = \epsilon$); the language $L \cdot L'$ is the set $\{uv \ / \ u \in L, v \in L'\}$; the language L^i is the set $\{u_1 u_2 \dots u_i, \ \forall j \ u_j \in L\}$ ($L^0 = \{\epsilon\}$); L^* is the union for $i \in \mathbb{N}$ of L^i.

We say that a word (*resp.* language) is *unary* (we also speak about *Tally* languages), if it is defined on a one-letter alphabet. The only interesting thing for a unary word is its length. Hence we will assimilate unary language with integer set (*i.e.*, the set of lenght of words from the unary language).

2.1.2. Finite automata

We give here some basic definitions in Automata Theory. We recall that a *finite automaton* (in its more general form) is a 5-tuple $(Q, \Sigma, q_0, F, \delta)$ where Σ is the input alphabet, Q is the finite set of states containing the *initial state* q_0 and the subset of *accepting states* F, and δ is the *transition function*. At each step, the automaton reads the symbol scanned by the input head, and thanks to its current state, it moves its input-head backward, forward or keep it in place, and change its state according to its transition function. For computation on an input word $w \in \Sigma^*$, the input tape contains $\dashv w \vdash$, where $\dashv$ and $\vdash$ (not belonging to Σ) are respectively the *left* and *right endmarkers*. We forbid the transition function to move the input head right (*resp.* left) from the right (*resp.* left) endmarker, hence the input head is not allowed to move out the input word.

A *configuration* (of an automaton on a word) is a couple (q, x) of $Q * \{0, \dots, n+1\}$ where n is the length of the input word, q is the current state and x is the current head position ($x = 0$ (*resp.* $n+1$) holds for the left (*resp.* right) endmarker). The *initial configuration* is $(q_0, 0)$, a *border configuration* is a configuration (q, p) where p is either 0 or $(n+1)$. An *accepting configuration* is a configuration $(q, n+1)$ with $q \in F$. From the transition function, we can define the relation $\rightarrow$ on configurations (note that the relation depends on the input word). $\rightarrow^*$ denote the transitive closure of $\rightarrow$. The automaton *accepts* a word w of size n if and only if $(q_0, 0) \rightarrow^* (q_f, n+1)$ for some $q_f \in F$. The *accepted language* is the set of all accepted words. A *computation* is a maximal[2] sequence of ($\rightarrow$)-successive configurations. A computation is said *accepting* if it contains an accepting configuration.

We distinguish several particular cases of automata:

- *deterministic/nondeterministic*: whether $\max_{q,c} |\delta(q, c)| \leq 1$
- *one-way/two-way*: whether backward moves of the input head are allowed
- *sweeping*: if the input head can change directions (forward/backward) only at the endmarkers. In that case we call *traversal* a computation path that starts from and ends by border configurations without encountring border between them.
- *unary*: if $|\Sigma| = 1$

In name of machines, we will use conventional letters or numbers: 1 (*resp.* 2) for one-way (*resp.* two-way), U for unary, D (*resp.* N) for deterministic (*resp.* nondeterministic), S for sweeping. The order is chosen by the author in order to make pronunciation easier, however 1-or-2 takes the first place while D-or-N are placed just before FA which is naturally always at the end, in order to save known structures on short names. For example, a $2SUNFA$ is a two-way sweeping unary nondeterministic finite automaton.

2.1.3. Known results

We give now some classical results. The first theorem is an old result (probably the oldest one) in Automata Theory. It answers the question of what is the computational power of FA model, while it caracterizes regular language class.

Theorem 1. *Finite Automata accepts exactly the class of regular languages.*

From this, it is easy to prove the famous Pumping Lemma:

Theorem 2 (Pumping Lemma)**.** *If a language L is regular, then there exists a constant N such that for every word w in L of length at least N, we can write $w = xyz$ (i.e., w can be divided into three substrings), satisfying the following conditions:*

- $|y| \geq 1$
- $|xy| \leq N$
- *for all i, xy^iz is in L.*

In the general case the converse of the lemma is not true, however, as said in introduction, the unary case has big differences. In fact for unary language, the converse turn out to be true.

[2] This sequence may be infinite, however one may force accepting computation to be finite by set transition from accepting configurations (position $n + 1$ is ensured by $\vdash$) to $\emptyset$.

Theorem 3. *A unary language L is regular if and only if it satisfies the Pumping Lemma.*

Another result on unary languages is the collapse between regular and context-free classes. It can be proved from the previous Theorem and an analog form of Pumping Lemma, for context-free languages.

Theorem 4. *Over one-letter alphabet, regular and context-free languages coincide.*

2.2. Finite Automata System

2.2.1. General definitions

We now present a parallel improvement on FAs. Several automata $\mathbf{A}[1], \ldots, \mathbf{A}[k]$ work on a same input tape. We want to give the possibility to each automaton to send and receive informations (*i.e.*, state). Suppose M is the message vector set (common for every automata), that we will describe below. For $\mathbf{m} \in M$, each coordinate $\mathbf{m}[i]$ corresponds to message sent by automaton $\mathbf{A}[i]$ (Nil, if no message is sent). $A = (Q, \Sigma, M, q_0, \delta, \nu)$ is a *k-communicating finite automaton* (k is the size of vectors of M) if δ is a function from $M * Q * (\Sigma \cup \{\dashv, \vdash\})$ into $\mathcal{P}(Q * \{-1, 0, +1\})$ and ν is a function from $Q * (\Sigma \cup \{\dashv, \vdash\})$ into[3] $\{0, 1\}$. If the communicating finite automaton is in state q with its input head scanning symbol c, then in a first time it decide using ν whether it sends a message or not: if ν return 1 then it sends message q, else it do not send message (*i.e.*, Nil). In a second time it receives message vector (of size k), and use δ to decide what state it enters and how it moves the input head.

We are now able to define *k-Communicating Finite Automata System* (FAS_k). A FAS_k is couple $(\mathbf{A}, F)$ where $\mathbf{A} = (\mathbf{A}[1], \ldots, \mathbf{A}[k])$ is a family of k k-communicating finite automata, and $F \subset \mathbf{Q}[1]$ is the set of accetping states ($\mathbf{Q}[1]$ is the state set of $\mathbf{A}[1]$, and more generally we use the notation $\mathbf{X}[i]$ for component X of $\mathbf{A}[i]$).

The message vector set M is equal to $\Pi_i(\mathbf{Q}[i] \cup \{Nil\})$ *i.e.*, its coordinates are either a state of the corresponding automaton or Nil. We designate by $\mathbf{Nil}$ the message vector where every coordinates are Nil.

A *global configuration* of a FAS_k on a word w of length n is a couple of vectors of size k $(\mathbf{q}, \mathbf{p})$, where $\mathbf{q}$ is the vector of state and $\mathbf{p}$ is the vector of positions (integers from $\{0, \ldots, n+1\}$). The *initial global configuration* is $\mathbf{c_0} = (\mathbf{q_0}, \mathbf{0})$. $(\mathbf{q}, \mathbf{p})$ is said *accepting* if $\mathbf{q}[1] \in F$ and $\mathbf{p}[1] = (n+1)$. As for simply case, we are able to deduce from a FAS a relation $\rightarrow$, depending on the word, such that $\mathbf{c} \rightarrow \mathbf{c}'$ if the system reaches global configuration $\mathbf{c}'$ from $\mathbf{c}$ in one step. $\rightarrow^*$ is the transitive closure of $\rightarrow$. The system accepts a word if $\mathbf{c_0} \rightarrow^* \mathbf{c}$ for some accepting global configuration $\mathbf{c}$. *Global computation* is defined as for

[3] Here we force ν to be deterministic, for more clarity. It is easy to see that, nondeterminism of δ may simulate nondeterminism of ν. Hence we make our assumption without loss of generality. A good question about the converse case (δ deterministic and ν nondeterministic) is raised in Section 4.

simply automaton *i.e.*, it is a sequence of ($\to$)-successive global configurations, starting from the initial one.

A *communicating step* (*resp.*, border step) is a step $\mathbf{c} \to \mathbf{c}'$ where the message vector $\mathbf{m}$ is different from **Nil** (*resp.*, at least one head is positionned on an endmarker). The *number of message* of a global computation is the sum over each step of the computation of the number of coordinates unequal to Nil, in exchanged message vectors. We say that a system has *communication complexity* Φ if for each accepted word w there exists an accepting computation which uses at most $\Phi(|w|)$ messages. In particular, we will study system with constant communication complexity.

2.2.2. One-way Unary Finite Automata System

Over one-letter alphabet, if each automaton component of a system is one-way (*i.e.*, the system is a $1UFAS$), then the communication complexity does not influence the computational power of the model. This holds even in both deterministic and nondeterministic cases.

This result follows a general result on unary multi-head FAs, proved by Ibarra and Harrison.

Corollary 1. *If a language is accepted by a* $1UFAS$*, then it is regular.*

So our subconjecture 1 on one-way systems is already solved. From now, we work only on two-way systems (omitting number "2" in short names).

2.3. Tri-phase Sweeping Unary Finite Automata System

Let us now focus on the deterministic case. First, observe behaviors of deterministic finite simple automata over unary input (*i.e.*, $UDFA$). Because the input alphabet contains only one letter, the input head can not observe differences between positions inside the word. Hence the most relevant steps in a computation are those that reach or leave a border configuration. Suppose we start computation from a border configuration (q_b, p_b) on a large enough[4] input w, and observer the $2 * |Q|$ following steps. There are three main cases:

1. either the input head is moved again to the border position p_b in less than $2 * |Q|$ steps,
2. or, it enters a deterministic loop in less than $|Q|$ steps *i.e.*, $(p_b, p_b) \to^{\leq |Q|} c_l \to^s c_l$ for some non-border configuration c_l and some constant s.
3. or the automaton enters a *state-loop* *i.e.*, the automaton reaches in less than $|Q|$ steps a configuration (q_l, p_l) such that $(q_l, p_l) \to^s (q_l, p'_l)$, for some s and $p'_l \neq p_l$.

Behavior (2) may be seen as a particular case of Behavior (3) with $p_l = p'_l$. However it is interesting to separate this behaviors, because behavior (2) does not allow the automaton to ever reach a border configuration again.

[4]By large enough, we mean that the length of the input is greater than $|Q| + 1$.

Between two border configurations (so behavior (2) cannot happen) behavior (1) gives an information of the form $|w| > m$ for some constant m bounded by $|Q|$, while behavior (3) gives an information on congruence of $|w|$ modulo the speed (also bounded by $|Q|$).

2.3.1. Tri-phase Sweeping Unary Deterministic Finite Automata

As describe above, the behavior of simple $UDFA$s may be easily described. Hence the work of such an automaton is well known. In the litterature we find simplification and normal forms for this model (see for example [**?**]). However in order to study $UDFA$ systems, we have to preserve the "speed of computation", because of synchronism. That is why we introduce here a new model, which can be seen as a normal form for $UDFA$s.

We define *Tri-phase Sweeping Unary Finite Automata* ($TSUDFA$ for short), which are deterministic sweeping automata over one-letter alphabet, that works for each traversal in three successive phases over large enough input:

- $Prefix$ *phase*: during this phase, the automaton move its input head in the same direction ($d \in \{-1, 0, +1\}$) at each step
- $Wait$ *phase*: during this phase, the automaton does not move its input head
- *Loop phase*: the automaton enters a state-loop, in which using d- (the same d as in $Prefix$ phase) and 0-moves, it moves at a "constant *speed*".

Formally such an automaton is defined using several state sets and a move function, as follow:

Definition 1. $(P, W, L_1, L_2, m, \Sigma, q_0, F, \delta)$ *is a* $TSUDFA$ *if and only if:*

- *P, W, L_1 and L_2 are disjoint finite state sets (let be $Q = P \cup W \cup L_1 \cup L_2$) and Σ is a single-letter alphabet (let us denote by 'a' its only symbol)*
- $((Q * \{-1, 0, +1\}), \Sigma, (q_0, -1), F, \delta)$ *is a unary 2DFA (we call* direction *the* $\{-1, 0, +1\}$ *state component)*
- *for each $q \in Q$, if $\delta((q, -1), \dashv)$ (resp. $\delta((q, +1), \vdash)$) is equal to $((q', d'), d)$ then $d' = d = +1$ (resp. -1) and $q' \in P$.*
- *for each $q \in P$ (resp. $q \in W$) there exists a finite state sequence $\{q_1, \ldots, q_{\pi+1}\}$ such that:*
  - $q_1 = q$
  - $\forall\ 1 \leq i \leq \pi$, $q_i \in P$ *(resp.* $q_i \in W$*)*
  - $q_{\pi+1} \in W$ *(resp.* $q_{\pi+1} \in L_1$*)*
  - $\forall\ 1 \leq i \leq \pi$, $\forall\ d \in \{-1, 0, +1\}$, $\delta((q_i, d), a) = ((q_{i+1}, d), d')$ *with* $d' = d$ *(resp.* $d' = 0$*).*

- *Let be $L = L_1 \cup L_2$. m is a function from L into $\{0,1\}$ and for each q in L there exists a finite state sequence $\{q_1, \dots, q_{\omega+1}\}$ in L such that:*
  - $q_1 = q_{\omega+1} = q$
  - $\forall\ 1 \leq i \leq \omega$, $\forall\ d \in \{-1, 0, +1\}$, $\delta((q_i, d), a) = ((q_{i+1}, d), d * m(q_i))$
  - $\exists\ 1 \leq i \leq \omega$, $q_i \in L_1$
  - *if* $q \in L_1$, *then for each* $1 \leq t \leq \omega$, $\sum_{i=1}^{t} m(q_i) = \left\lceil t * \frac{\sum_{i=1}^{\omega} m(q_i)}{\omega} \right\rceil$.

In computation of such a model, the input head works in a sweeping manier, because of the direction component and reversals at the endmarkers. Remark first that the assumption about endmarkers transition enforces the input head to leave an endmarker in at most one step. Observe also that deterministic loops are not forbidden (just see the case $m(q_i) = 0$ for each i in a state sequence of L, as in Definition). Hence the automaton, starting from a configuration, has two types of behavior:

1. either it does a (possibly partial) traversal of the input word, until it reaches an endmarker (with all move of the input head in the same direction, given by the direction component)
2. or, after a constant number of steps, it loops inside the input word, without moving its input head.

More precisely, from each configuration, after at most $|P| + |W|$ steps, if the input head has not reached an endmarker, the automaton enters a state-loop of fixed period size (at most $|L|$). In Definition, the last condition on state sequence starting from $q \in L_1$, ensure us to have some kind of *constant speed* (see variable t as a number of steps since $\frac{1}{\omega} * \sum_{i=1}^{\omega} m(q_i)$ is the speed average over one state-period). This *speed* may be equal to 0, in case of real-loop (behavior (2) described above). The goal of the following Lemma is to describe these behaviors (and define the notion of speed).

Lemma 1. *Let $(P, W, L_1, L_2, \Sigma, q_0, F, \delta)$ be a $TSUDFA$ and let w be a word of length n. For each state (q, d), there exist positive integers $\pi_1 < \pi_2$ and $\omega > \Delta$ such that for every head position p on w ($0 \leq p \leq (n+1)$) and for all $s \in \mathbb{N}$, if $\mathcal{A}$ performs s steps from Configuration $((q, d), p)$ without encountring an endmarker then*

- *if $s < \pi_1$ then the input head is in position $p + s * d$.*
- *if $\pi_1 \leq s < \pi_2$ then the input head is in position $p + \pi_1 * d$.*
- *if $\pi_2 \leq s$ then the input head is in position $p + (\pi_1 + \lceil (s - \pi_2) * \frac{\Delta}{\omega} \rceil) * d$.*

We call *speed* the rational $\frac{\Delta}{\omega}$, and for each $x \in \{\pi_1, \pi_2, \Delta, \omega\}$ and $q \in Q$ and $d \in \{-1, 0, +1\}$, we use the notation $x(q, d)$ to refer the corresponding parameter.

Proof. The Lemma statements directly result from Definition 1. See π_1 as the length of the $Prefix$ phase sequence, in fact exactly the "π" for sequence starting in $q \in P$ from Definition (0 if $q \notin P$) and π_2 as the length of both $Prefix$ and $Wait$ phase sequences, that is the sum of two "πs", as in Definition, for two connected ($Prefix$ and $Wait$) sequences starting in q (0 if $q \notin P \cup W$). Consider ω as the period of the state-loop sequence (the minimal "ω" in Definition) and Δ as the number of move (in direction d) in exactly one such period (that is "$\sum_{i=1}^{\omega} m(q_i)$" in Definition).

Therefore depending on which phase the automaton is performing after s steps (and supposing it does not reach an endmarker during these steps), we obtain respectively the three statements of the Lemma. $\square$

From this Lemma, we are now able to compute from each configuration $c = ((q,d),p)$ the number $nextborder(c)$ of steps needed by the automaton in order to reach the next endmarker (in case of deterministic loop, we set it to $+\infty$). In fact, in the case where $p + \pi_1 * d \leq 0$ (*resp.*, $p + \pi_1 * d \geq n+1$) it is easy to see that $nextborder(c)$ is equal to the minimal s in $\{1, \ldots, \pi_1\}$ such that $p + s * d$ is equal to 0 (*resp.*, $n+1$). In the other case, it can be found by solving the following equation:

$$p + (\pi_1 + \left\lceil (s - \pi_2) * \frac{\Delta}{\omega} \right\rceil) * d = \begin{cases} 0 \; if \; d = -1 \\ n+1 \; if \; d = +1 \end{cases}$$

Thus we obtain the following corollary.

Corollary 2. *For each state (q,d), one can find a rational α and an integer β such that for each head position p, supposing $d = +1$ (resp. -1) if $(n-p)$ (resp. p) is larger than $\pi_1(q,d)$, then $nextborder((q,d),p)$ is equal to $\alpha * (n-p) + \beta$ (resp. $\alpha * p + \beta$).*

Proof. The proof is a simply solve of the previous equation. $\square$

The following remark is a particular case of this corollary (and first case, where $p + \pi_1 * d$ is smaller than 0 or greater than $(n+1)$):

Remark 1. *If p is an affine function of n, then $nextborder(q,d)$ may also be exprimed as an affine expression of n.*

2.3.2. Tri-phase Sweeping Unary Deterministic Finite Automata Systems

We now define *$TSUDFA$ System*, in which each automaton component is a $TSUDFA$ (with transition function $\boldsymbol{\delta}[i]$ considered with communication vector **Nil**). In particular, each component may change its direction only if its heads is reading an endmarker or if a message is sent ($\mathbf{M} \neq Nil$). Without loss of generality, we suppose that for each automaton $\mathbf{A}[i]$, each state q of $Q[i]$, each input symbol x and each communication vector $\mathbf{M} \neq \mathbf{Nil}$, $\boldsymbol{\delta}[i](q, \mathbf{M}, x) = (q', d')$ implies that $q' \in \mathbf{P}[i]$ (this can be done by adding two copies of each state not in $\mathbf{P}[i]$, in $\mathbf{P}[i]$ and $\mathbf{W}[i]$).

Let us fix a $TSUDFAS$ S accepting a language $\mathcal{L}$. On computation over a input word w, we consider *communication events* and *border events*, that designate respectively when communication or border step occur. Between two successive such events, the behavior of each $TSUDFA$ component is deterministic and described previously. Hence we search now to describe the configuration of the system at a *particular event* (*i.e.*, communication or border event), in function of the configuration of the system at the previous one.

Lemma 2. *In each computation, the number of steps between two successive communication events is bounded by some function affine in n.*

Proof. Using the fact that at least one automaton does not loop between two successive particular event (at least one automaton will send a message in the next communication event), this result is a direct consequence of Corollary 2. □

From this Lemma, one may easily prove the following corollary:

Corollary 3. *In every computation, between two successive communication events, there are a bounded number of border events.*

As a particular case of this result, the following corollary is one of the key point used in Section 3.

Corollary 4. *If the system has a constant communication complexity, then the total number of particular events is also bounded by a constant.*

3. Main Result

Our goal is now to prove, in a first time, that $TSUDFA$ system with a constant number of communication accepts only regular languages. In a second time we will prove that every $2UDFAS$ can be simulated by a $TSUDFAS$ with a linear increase in the number of communication. These two points directly imply the following theorem:

Theorem 5. *Every language accepted by a $2UDFAS$ with a constant number of communication is regular.*

3.1. $TSUDFAS$ with a constant number of communications

Theorem 6. *If a language $\mathcal{L}$ is accepted by a $TSUDFAS$ with a constant communication complexity, then $\mathcal{L}$ is regular.*

Proof. Let $(\mathbf{A}, F)$ be a $TSUDFAS_k$ accepting a language $\mathcal{L}$. Suppose $(\mathbf{A}, F)$ has constant communication complexity *i.e.*, there exists a constant C such that for each input word, computation (recall $TSUDFAS$s are deterministic machines) uses at most C messages (one may suppose that a counter of messages is saved in state information, in order to force every computation to use exactly C messages).

We will find a finite regular partition $\mathcal{R}$ of Σ^* such that knowing that a word w belongs to some regular language L of $\mathcal{R}$, one can find a Presburger Formula depending only on the size n of w (the only free variable), such that the formula is true if and only if w is accepted by $(\mathbf{A}, F)$ *i.e.*, $w \in \mathcal{L}$. By Theorem **??**, this implies that $\mathcal{L} \cap L$ is regular. Hence, because regular class is closed under finite union, $\bigcup_{L \in \mathcal{R}} (\mathcal{L} \cap L)$ is regular. Finally, this will be implies that $\mathcal{L}$ is regular, because $\mathcal{R}$ is a partition of Σ^*.

By Corollary 4, there is a finite number $T \geq C$ such that in each accepting computation there are at most T particular events. Without loss of generality suppose that there are exactly T particular events in every accepting computations (one may enforce this property by sending message at each particular event, counting them and add some messages at the end if necessary).

First, we prove the following lemma:

Lemma 3. *There exists a regular partition $\{L_1, L_2, \ldots, L_\Phi\}$ of Σ^* such that, for each $i \in \{1, \ldots, k\}$ and $t \in \{1, \ldots, T\}$ there are computable functions:*

- $\boldsymbol{\alpha}^t[i]$ *from* $\{1, \ldots, \Phi\}$ *to* $\mathbb{Q} \cap [0, 1]$
- $\boldsymbol{\beta}^t[i]$ *from* $\{1, \ldots, \Phi\}$ *to* $\mathbb{Z} \cap [(|Q| * t), (|Q| * t)]$
- $\boldsymbol{\gamma}^t[i]$ *from* $\{1, \ldots, \Phi\}$ *to* Q

such that for each $w \in \mathcal{L}$ of size n, $w \in L_j$ implies that when the t-th particular event occurs automaton $\mathbf{A}[i]$ is in state $\boldsymbol{\gamma}^t[i](j)$ with its input head reading the $(\mathbf{p}^t[i](j) = \boldsymbol{\alpha}^t[i](j) * (n+1) + \boldsymbol{\beta}^t[i](j))$-th symbol of the input (so in particular $\mathbf{p}^t[i](j)$ has to be an integer of the interval $[0; n+1]$).

Proof. In order to prove this Lemma, we prove by induction on $1 \leq \tau \leq T$ that there are Φ^τ, $\mathcal{R}^\tau = \{L_1^\tau, \ldots, L_{\Phi^\tau}^\tau\}$ and functions $(\boldsymbol{\alpha}^\tau[i])_{1 \leq i \leq k}$, $(\boldsymbol{\beta}^\tau[i])_{1 \leq i \leq k}$ and $(\boldsymbol{\gamma}^\tau[i])_{1 \leq i \leq k}$ satisfying the Lemma statement.

If $\tau = 1$, we just have to look at the initial configuration, which is the first particular event. So, according to the definition, we can set $\Phi^1 = 1$, $\mathcal{R}^1 = \{\Sigma^*\}$, and for each i: $\boldsymbol{\alpha}[i]^1(1) = \boldsymbol{\beta}[i]^1(1) = 0$ and $\boldsymbol{\gamma}[i]^1(1) = \mathbf{q_0}[i]$. Trivialy these partition and functions satisify the Lemma statement.

Suppose now that for $1 \leq \tau < T$, we have Φ^τ, $\mathcal{R}^\tau = \{L_1^\tau, \ldots, L_{\Phi^\tau}^\tau\}$ and functions $(\boldsymbol{\alpha}^\tau[i])_{1 \leq i \leq k}$, $(\boldsymbol{\beta}^\tau[i])_{1 \leq i \leq k}$ and $(\boldsymbol{\gamma}^\tau[i])_{1 \leq i \leq k}$ satisfying the statement of the lemma. Let be $w \in \mathcal{L}$ of size n. Let j be such that $w \in L_j^\tau$. Let $\mathbf{c}^\tau$ be the configuration of the system when the τ-th event occurs. Positions (*resp.* states) of the automata in $\mathbf{c}^\tau$ are given by $(\mathbf{p}^\tau[i](j) = \boldsymbol{\alpha}^\tau[i](j) * n + \boldsymbol{\beta}^\tau[i](j))_{1 \leq i \leq k}$ (*resp.* $(\boldsymbol{\gamma}^\tau[i](j))_{1 \leq i \leq k}$).

Without loss of generality, we suppose that n is large enough to ensure that for each i, $\mathbf{p}^\tau[i](j)$ (*resp.* $n - \mathbf{p}^\tau[i](j)$) is less than $|Q|$ implie that $\boldsymbol{\alpha}^\tau[i](j)$ is equal to 0 (*resp.* 1).

Now we look at the successor configuration of $\mathbf{c}^\tau$, called $\mathbf{c}^{s(\tau)}$. Observe first that we can compute it from $(\mathbf{p}^\tau[i])_i$ and $(\boldsymbol{\gamma}^\tau[i])_i$, and we can find $\boldsymbol{\gamma}^{s(\tau)}[i]$ and

$\boldsymbol{\beta}^{s(\tau)}[i]$ ($\boldsymbol{\alpha}^{s(\tau)}[i] = \boldsymbol{\alpha}^{\tau}[i]$) such that $\mathbf{p}^{s(\tau}[i] = \boldsymbol{\alpha}^{s(\tau)}[i] * (n+1) + \boldsymbol{\beta}^{s(\tau)}[i]$ is the position of automaton $\mathbf{A}[i]$ in configuration $\mathbf{c}^{s(\tau)}$ and $\boldsymbol{\gamma}^{s(\tau)}[i]$ is its state. There are two possible cases:

- If $\mathbf{c}^{s(\tau)}$ is a border or communicating configuration, so we have already reached the next particular event, then we can conserve the same regular partition ($L_i^{\tau+1} = L_i^{\tau}$), and set $X_i^{\tau+1}$ to $X_i^{s(\tau)}$ for $X \in \{\alpha, \beta, \gamma\}$.

- Else, from this configuration each automaton works independently until the next particular event. (This first step (from $\mathbf{c}^{\tau}$ to $\mathbf{c}^{s(\tau)}$) ensure that each automaton already took into account the possible messages of the τ-th particular event.)

  Let be $E \subset \{1, \ldots, k\} * \mathbb{N}$ such that (i, x) is in E if and only if for every large enough input word $w \in L_j$ of size n, starting with head positionned in $\boldsymbol{\alpha}^{s(\tau)}[i] * (n+1) + \boldsymbol{\beta}^{s(\tau)}[i]$ and state $\boldsymbol{\gamma}^{s(\tau)}[i]$, supposing no messages are sent by others automata, automaton $\mathbf{A}[i]$, reaches in x steps a border or communicating configuration for the first time. Observe that x has to be the same for every large enough input word of L_j (so it depends only on j).

  The set E is the set of indices of automata of the system, which, starting in configuration $\mathbf{c}^{s(\tau)}$ may provocate a particular event in a number of steps non depending on n. This set is trivially computable from $(\boldsymbol{\alpha}^{\tau}[i])_i$, $(\boldsymbol{\beta}^{\tau}[i])_i$ and $(\boldsymbol{\gamma}^{\tau}[i])_i$ by a simple simulation of local machines.

  Suppose $E \neq \emptyset$. Then we can find the minimal x, such that there is i_0 (non necessary unique) such that (i_0, x) is in E. This means that, over large enough input, starting from $\mathbf{c}^{\tau}$, the system, after having performed exactly $(x+1)$ steps enters the $(\tau+1)$-th particular event, which is provocated by (at least) Automaton $\mathbf{A}[i_0]$.

  We can find for each automaton $\mathbf{A}[i]$, the state q_i it enters after x local steps starting from $\mathbf{c}^{s(\tau)}$, and the length Δ_i and direction d_i of the corresponding head move. Both values are independent on n, because x is a constant.

  Hence, we can set:

  - $\Phi^{\tau+1} = \Phi^{\tau}$
  - $\mathcal{R}^{\tau+1} = \mathcal{R}^{\tau}$
  - $\forall\ i\ \boldsymbol{\alpha}^{\tau+1}[i] = \boldsymbol{\alpha}^{\tau}[i]$
  - $\forall\ i\ \boldsymbol{\beta}^{\tau+1}[i] = \boldsymbol{\beta}^{\tau}[i] + \Delta_i * d_i$
  - $\forall\ i\ \boldsymbol{\gamma}^{\tau+1}[i] = q_i$

  which satisfies the statement at rank $\tau + 1$.

  In the last case ($E = \emptyset$), the $(\tau+1)$-th particular event happens after more than $|Q|$ steps. So, each automaton enters in a state-loop. Hence according to Remark **??**, there exist for each i, two constant of same sign: σ_i and

μ_i, such that if the automaton $\mathbf{A}[i]$ don't receive messages, it reaches the next endmarker in exactly $\sigma_i*(n+1)+\mu_i$ steps. Hence, there exists j such that, for large enough n, A_j is (one of) the first automaton to reach the endmarker. Hence we know $s^\tau = \sigma_i*(n+1)+\mu_i+1$, the number of steps required to perform computation part between particular events τ and $(\tau+1)$. Then, according to Lemma 1, for each automaton $\mathbf{A}[i]$ the position $\mathbf{p}^{\tau+1}[i]$ of the input head can be computed from three constants $\pi_1 < \pi_2 < |Q| \in \mathbb{N}$ and $v \in \mathbb{Q} \cap [0,1]$ by $\mathbf{p}^{\tau+1}[i] = \mathbf{p}^\tau[i] + \pi_1 + \lfloor v*(s^\tau - \pi_2)\rfloor$ and the state depends only on the $s^\tau \mod l$, for some known constant l.

So, using the linear expression of s^τ, one can find a regular finite partition which give us the required information to compute position and state of automata at $(\tau+1)-th$ particular event. Thus, by doing intersection of this partition and $\mathcal{R}^\tau$, we obtain a new finite regular partition $\mathcal{R}^{\tau+1} = \{L_1, \ldots, L_{\Phi^{\tau+1}}\}$ for which we can compute functions $(\boldsymbol{\alpha}^{\tau+1}[i])_{1\le i\le k}$, $(\boldsymbol{\beta}^{\tau+1}[i])_{1\le i\le k}$ and $(\boldsymbol{\gamma}^{\tau+1}[i])_{1\le i\le k}$ such that:

$$w \in \mathcal{L} \cap L_j \Rightarrow \quad \left\{ \begin{array}{c} \forall\ 1 \le i \le k, \\ \textit{when the } (\tau+1)-th \textit{ particular event occurs}, \\ \mathbf{A}[i] \textit{ has its input head in position}: \\ \boldsymbol{\alpha}^{\tau+1}[i](j)*(n+1) + \boldsymbol{\beta}^{\tau+1}[i](j) \\ \textit{and its state is } \boldsymbol{\gamma}^{\tau+1}[i](j) \end{array} \right.$$

This concludes our induction and therefore the proof of Lemma 3.

□

So, now we have from previous lemma, a regular partition $\mathcal{R}$ (of size Φ), and functions $(X^t[i])_{\substack{1\le i\le k\\ 1\le t\le T}}$ for each $X \in \{\alpha, \beta, \gamma\}$, which describe every particular configurations in accepting computation.

Observe that because $\mathcal{R}$ is finite and T is constant, we have a finite number of parameters. So we can compute and save them in a three dimensional matrix. By multipliying each parameters by a constant, we may work only with integers.

Suppose w (of size n) belongs to some regular language L of partition $\mathcal{R}$. w is accepted by $(\mathbf{A}, F)$ if and only if $\boldsymbol{\alpha}^T[1]*n + \boldsymbol{\beta}^T[1] = n+1$ and $\boldsymbol{\gamma}^T[1] \in F$. Finally $\mathcal{L}$ is equal to the union of languages L_j from $\mathcal{R}$, such that $\boldsymbol{\alpha}^T[1](j) = 1$, $\boldsymbol{\beta}^T[1](j) = 1$ and $\boldsymbol{\gamma}^T[1](j) \in F$. Thus $\mathcal{L}$ is a finite union of regular languages, so $\mathcal{L}$ is regular. □

3.2. 2DFAS simulation by TSUDFAS

In order to prove Theorem **??**, we have to prove that each $2UDFAS$ with constant communication complexity may be simulated by an equivalent $TSUDFAS$ with constant communication complexity. This is directly implied by the following theorem.

Theorem 7. *For each $2UDFAS_k$ with communication complexity $f(n)$ there is an equivalent $TSUDFAS_k$ with communication complexity $g(n)$, $f(n) = \mathcal{O}(g(n))$.*

To avoid technical details, we give here only main ideas of the proof ; please look at figures.

Proof. Let us fix a $2UDFAS_k$ $(\mathbf{A}, F)$. Suppose it has constant communication complexity. Recall that, because the alphabet has only one letter, the local behaviors over large enough input words are simple (see description in Section **??**).

We have to transform locally each component in order to make it sweeping and three-phase.

Observe behavior (3) (from description of Section **??**). There already exists two phases, one prefix and one state-loop (if prefix does not exist, we can duplicate states of the loop to simulate prefix phase, whitout changing information and computational speed).

- First we change the move order in prefix phase (see Figure 1). We start traversal with all steps which move the input head. Then we stay in place, waiting for the time of the end of prefix phase. Thus the input head is always in advance compared with original computation, and it never perform backward moves. We have created our $Prefix$ and $Wait$ phases, like in $TSUDFA$ definition (see Definition 1). At each step, the original state information is saved in state (*i.e.*, we just change moves).
- In a second time we modify the state-loop, in order to eliminate backward moves. See on Figure 1 the asymptotic line to the head move. We want to follow this line, with same rate, approaching the original moves by excess, in order to have the simulating head always in advance according to the original one. This will create the *Loop* phase of $TSUDFA$s.

Behavior (2) may be simulated with the same method. For this case, the loop has to be in place, at the extremal position of the head (see Figure 2).

The last behavior (*i.e.*, Behavior (1)), has to be recursively simulated. In less than $2*|Q|$ steps, the head is reading the initial endmarker again. Hence we study the Hence, in at most $2*|Q|*|Q|$, the automaton has entered a different behavior (*i.e.*, (3) or (2)), or it has entering a deterministic loop, which rebounds on the endmarker. These two cases are treated separately.

For the first case, we can simulate the new behavior, inserting in wait phase a lot (but a constant number) of states, which will give us the possibility to wait for the good time for start state-loop (see Figure 3). For the second case, because we want to forbid to visits to many times an endmarker (see Lemma **??**), we have to loop, on place (because of sweeping), inside the word (see Figure 4).

After all these transformations, at each step, the each simulating component knows the state of the simulated component, and its input head is positionned at a constant distance of the simulated head. Hence in a constant bounded number of steps, each component is able to retrieve the configuration of the original machine.

Now, we have to take into account the messages. This can be done by the same method, however if message are received in midle of input (not at endmarkers), backward moves may be important, because they can test that an endmarker is far enough from the current position (see Figure **??**).

That is why we simulate this part of computation in two times, sending another message after performed a constant number of steps after each communicating event. (So here, we add some steps, and we multiply by two the number of messages.)

By construction, the simulating machine accepts the same language as the initial $UDFAS$. This concludes our proof. □

4. Conclusion

In this 6-months internship, I solved a part of our initial conjecture: Two-way Communicating Unary Deterministic Finite Automata Systems accepts only regular languages. The nondeterministic case remains unsolved. We have tried several approaches to solve the problem, like generalisation or particular case.

The proof of Section **??**, comes from the second approach (*i.e.*, we look first at pseudo-sweeping deterministic finite automata systems, and then we find the second part of the proof (Section **??**) that finished the proof of Theorem **??**).

Generalisation turn out to be very difficult, because the work area is very tight: we know that for non unary alphabet the conjecture is false[5]. If we try to increase the number of messages, we know that there is a $2UDFAS_2$ which accepts nonregular language $\left\{1^{2^n} \ / \ n \in \mathbb{N}\right\}$. We conjecture that there exists a gap between constant and logarithmic communication complexity.

In order to solve the conjecture in nondeterministic case, we will try to limit nondeterminism to communication function ν. Thus, between two messages, each automaton component of the system will have a pseudo-deterministic behavior (in fact, if an automaton does not receive a message, it "knows" how others are working). This is an interesting question, but we do not have any result about this particular case for the moment.

This internship has increased my experience in Automata Theory, and more generally in Research Work. I had the chance to participate to the french *École Jeunes Chercheurs en Informatique et Mathématiques* where I presented results from the previous internship (so I did my first presentation in an official work shop).

I knew to kind of difficulties during the internship. I had difficulties to find ideas to approach our main problem. That is why I read (or take a look) at many papers (see Bibliography). I had also a lot of difficulties to formally write proof and to write this report, because of abondance of technical details, particular cases and because of time.

[5] Even with only one occurence of a different symbol in each word of a language, $2UFAS$s will accept nonregular languages (see for example $\{1^n\#1^m\}$ which is analog to a^nb^n). I studied this kind of languages during my internship. Other more strange languages may be accepted, for example $\{1^n\#1^m \ / \ \gcd(n,m) = 1\}$.

I will continue to work in the subject, and try to solve the Conjecture 1 in its general form.

Bibliography

[1]

[2] *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS-01), San Francisco, CA, April 23-27, 2001*. IEEE Computer Society, 2001.

[3] Manindra Agrawal and Anil Seth, editors. *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings*, volume 2556 of *Lecture Notes in Computer Science*. Springer, 2002.

[4] Giorgio Ausiello and Corrado Böhm, editors. *Automata, Languages and Programming, Fifth Colloquium, Udine, Italy, July 17-21, 1978, Proceedings*, volume 62 of *Lecture Notes in Computer Science*. Springer, 1978.

[5] Piotr Berman. Relationship between density and deterministic complexity of np-complete languages. In *ICALP*, pages 63–71, 1978.

[6] Alberto Bertoni, Giancarlo Mauri, and Mauro Torelli. Some recursive unsolvable problems relating to isolated cutpoints in probabilistic automata. In *ICALP*, pages 87–94, 1977.

[7] Lothar Budach, editor. *Fundamentals of Computation Theory, 8th International Symposium, FCT '91, Gosen, Germany, September 9-13, 1991, Proceedings*, volume 529 of *Lecture Notes in Computer Science*. Springer, 1991.

[8] Marek Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.

[9] Tyng-Ruey Chuang and Benjamin Goldberg. Real-time deques, multihead thring machines, and purely functional programming. In *FPCA*, pages 289–298, 1993.

[10] Michal Chytil, Ladislav Janiga, and Václav Koubek, editors. *Mathematical Foundations of Computer Science 1988, MFCS'88, Carlsbad, Czechoslovakia, August 29 - September 2, 1988, Proceedings*, volume 324 of *Lecture Notes in Computer Science*. Springer, 1988.

[11] Pavol Duris and Juraj Hromkovic. Multihead finite state automata and concatenation. In *ICALP*, pages 176–186, 1982.

[12] Pavol Duris, Tomasz Jurdzinski, Miroslaw Kutylowski, and Krzysztof Lorys. Power of cooperation and multihead finite systems. In *ICALP*, pages 896–907, 1998.

[13] Shimon Even and Oded Kariv, editors. *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *Lecture Notes in Computer Science*. Springer, 1981.

[14] Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FSTTCS*, pages 145–156, 2002.

[15] Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors. *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*. ACM, 1979.

[16] Rusins Freivalds. Projections of languages recognizable by probabilistic and alternating finite multitape automata. *Inf. Process. Lett.*, 13(4/5):195–198, 1981.

[17] Laurent Fribourg and Hans Olsén. Proving safety properties of infinite state systems by compilation into presburger arithmetic. In *CONCUR*, pages 213–227, 1997.

[18] Thomas Gazagnaire. *Langages de scénarios : Utiliser des ordres partiels pour modéliser, vérifier et superviser des systèmes parallèles et répartis.* These, Université Rennes 1, 2008.

[19] Dainis Geidmanis. On possibilities of one-way synchronized and alternating automata. In *MFCS*, pages 292–299, 1990.

[20] Seymour Ginsburg, H. Gordon Rice, and H. Gordon Rice. Two families of languages related to algol. pages 350–371, 1962.

[21] Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[22] Eitan M. Gurari and Oscar H. Ibarra. The complexity of the equivalence problem for counter machines, semilinear sets, and simple programs. In *STOC*, pages 142–152, 1979.

[23] Yo-Sub Han, Kai Salomaa, and Derick Wood. Prime decompositions of regular languages. In *Developments in Language Theory*, pages 145–155, 2006.

[24] Michael A. Harrison and Oscar H. Ibarra. Multi-tape and multi-head pushdown automata. *Information and Control*, 13(5):433–470, 1968.

[25] J. Hartmanis. *On non-determinacy in simple computing devices.* Seminarberichte des Instituts für Theorie der Automaten und Schaltnetzwerke. GMD, 1971.

[26] Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-head finite automata: Characterizations, concepts and open problems. In *CSP*, pages 93–107, 2008.

[27] Markus Holzer, Martin Kutrib, and Giovanni Pighizzini, editors. *Descriptional Complexity of Formal Systems - 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25-27, 2011. Proceedings*, volume 6808 of *Lecture Notes in Computer Science*. Springer, 2011.

[28] Juraj Hromkovič and Georg Schnitger. Lower bounds on the size of sweeping automata. *J. Autom. Lang. Comb.*, 14(1):23–31, January 2009.

[29] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25(1):116–133, 1978.

[30] Oscar H. Ibarra and Zhe Dang, editors. *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*. Springer, 2006.

[31] Katsushi Inoue, Itsuo Takanami, Akira Nakamura, and Tadashi Ae. One-way simple multihead finite automata. *Theor. Comput. Sci.*, 9:311–328, 1979.

[32] Tomasz Jurdzinski. *Communication Aspects of Computation of Systems of Finite Automata*. PhD thesis, 1999.

[33] Tomasz Jurdzinski and Miroslaw Kutylowski. Communication gap for finite memory devices. In *ICALP*, pages 1052–1064, 2001.

[34] Tomasz Jurdzinski, Miroslaw Kutylowski, and Jan Zatopianski. Communication complexity for asynchronous systems of finite devices. In *IPDPS*, page 139, 2001.

[35] Janis Kaneps. Regularity of one-letter languages acceptable by 2-way finite probabilistic automata. In *FCT*, pages 287–296, 1991.

[36] K. N. King. Alternating multihead finite automata (extended abstract). In *ICALP*, pages 506–520, 1981.

[37] Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating pushdown and stack automata. *SIAM J. Comput.*, 13(1):135–155, 1984.

[38] Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors. *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*. Springer, 1998.

[39] Antoni W. Mazurkiewicz and Józef Winkowski, editors. *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*. Springer, 1997.

[40] Carlo Mereghetti and Giovanni Pighizzini. Optimal simulations between unary automata. In *STACS*, pages 139–149, 1998.

[41] Pascal Michel. An np-complete language accepted in linear time by a one-tape turing machine. *Theor. Comput. Sci.*, 85(1):205–212, August 1991.

[42] Burkhard Monien. Two-way multihead automata over a one-letter alphabet. *ITA*, 14(1):67–82, 1980.

[43] Michel Morvan, Christoph Meinel, and Daniel Krob, editors. *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*. Springer, 1998.

[44] Turlough Neary, Damien Woods, Anthony Karel Seda, and Niall Murphy, editors. *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008*, volume 1 of *EPTCS*, 2009.

[45] Mogens Nielsen and Branislav Rovan, editors. *Mathematical Foundations of Computer Science 2000, 25th International Symposium, MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000, Proceedings*, volume 1893 of *Lecture Notes in Computer Science*. Springer, 2000.

[46] Mogens Nielsen and Erik Meineche Schmidt, editors. *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, volume 140 of *Lecture Notes in Computer Science*. Springer, 1982.

[47] Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors. *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*. Springer, 2001.

[48] Rohit Parikh. On context-free languages. pages 570–581, 1966.

[49] H. Petersen. Alternation in simple devices. In *Proceedings of the 22nd International Colloquium on Automata, Languages and Programming*, ICALP '95, pages 315–323, London, UK, UK, 1995. Springer-Verlag.

[50] Giovanni Pighizzini. Unary pushdown automata and auxiliary space lower bounds. In *MFCS*, pages 599–608, 2000.

[51] Giovanni Pighizzini. Deterministic pushdown automata and unary languages. *Int. J. Found. Comput. Sci.*, 20(4):629–645, 2009.

[52] Giovanni Pighizzini. Nondeterministic one-tape off-line turing machines and their time complexity. *CoRR*, abs/0905.1271, 2009.

[53] Branislav Rovan, editor. *Mathematical Foundations of Computer Science 1990, MFCS'90, Banská Bystrica, Czechoslovakia, August 27-31, 1990, Proceedings*, volume 452 of *Lecture Notes in Computer Science*. Springer, 1990.

[54] Arto Salomaa and Magnus Steinby, editors. *Automata, Languages and Programming, Fourth Colloquium, University of Turku, Finland, July 18-22, 1977, Proceedings*, volume 52 of *Lecture Notes in Computer Science*. Springer, 1977.

[55] Joel I. Seiferas. Techniques for separating space complexity classes. *J. Comput. Syst. Sci.*, 14(1):73–99, 1977.

[56] Anna Slobodová. On the power of communication in alternating machines. In *MFCS*, pages 518–528, 1988.

[57] K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics Application.

[58] Jirí Wiedermann. Complexity of nondeterministic multitape computations based on crossing sequences. In *DCFS*, pages 314–327, 2011.

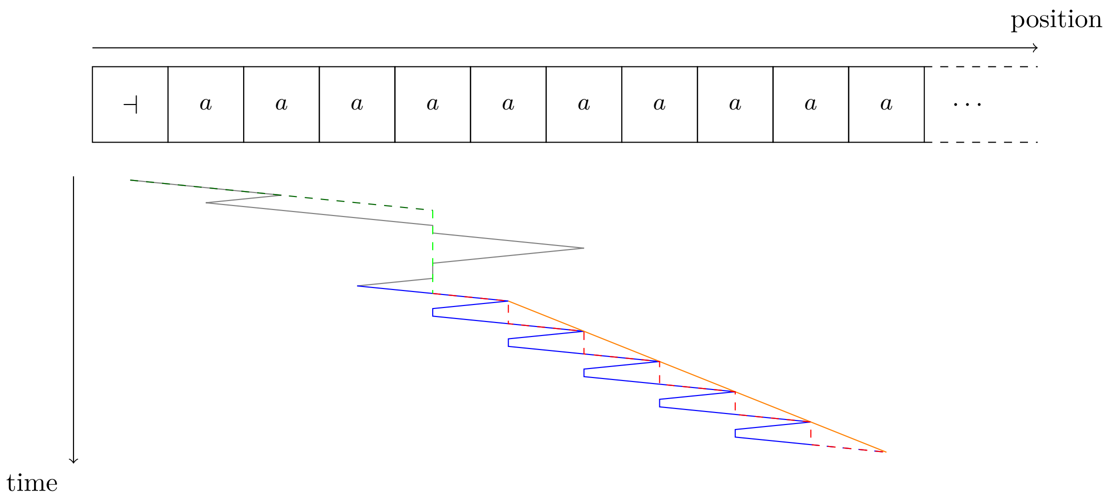

Figure 1: Behavior (3) (from left endmarker) simulated in a sweeping manier
in gray: initial prefix phase,
in blue: initial state-loop phase,
in dashed dark green: simulating *Prefix* phase,
in dashed green: simulating *Wait* phase,
in dashed red: simulating *Loop* phase,
and in orange: asymptotic speed.

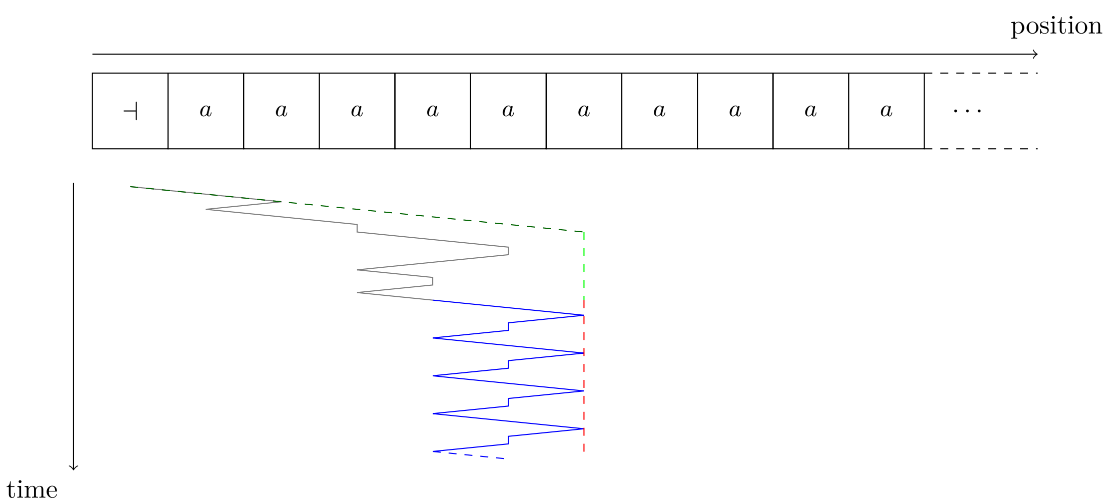

Figure 2: Behavior (2) simulated in a sweeping three phase manier
in gray: initial prefix phase,
in blue: initial loop,
in dashed dark green: simulating *Prefix* phase,
in dashed green: simulating *Wait* phase
and in dashed red: simulating *Loop* phase.

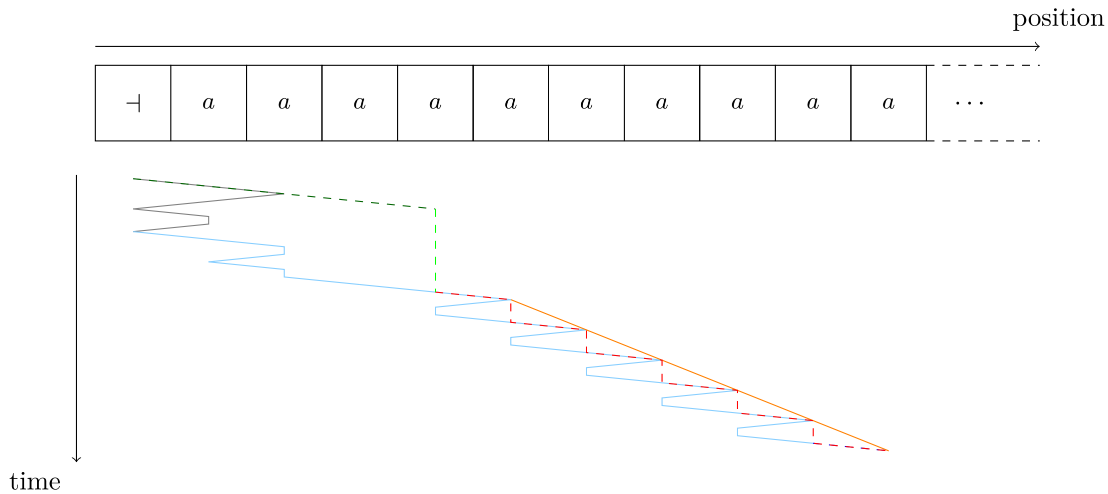

Figure 3: Simulation of Behavior (1)
in gray: initial Behavior (1) (two times),
in light blue: initial Behavior (3),
in dashed dark green: simulating $Prefix$ phase,
in dashed green: simulating $Wait$ phase,
in dashed red: simulating *Loop* phase.

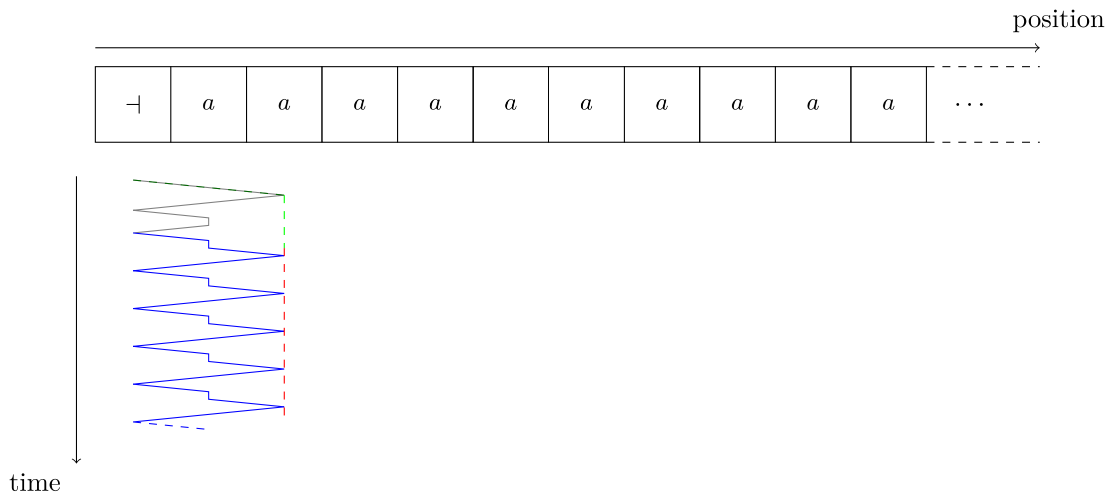

Figure 4: Simulation of Behavior (1)
in gray: initial Behavior (1) in a loop,
in dashed dark green: simulating $Prefix$ phase,
in dashed green: simulating $Wait$ phase,
in dashed red: simulating *Loop* phase.