# Lecture notes on regular cost functions (MPRI 2.16, Modèles de calcul et automates finis)

Thomas Colcombet

MPRI 2024-2025

These notes were last compiled on January 8, 2025.
These notes are in permanent rewriting, and are likely to contain errors and typos (you can report them if you want). They make use of internal hyperlinks, and may be better read on an electronic support.
More complete information can be found at
`https://www.irif.fr/~colcombe/Publications/habilitation-colcombet_v1.1.pdf`.

# Table of Contents

## Introduction

This part of the course describes the notion of a regular cost function, a quantitative extension of the notion of regular word languages. The principle of this theory is to study not languages, i.e. applications of words in $\{\text{inside}, \text{outside}\}$, but functions in $\omega+1$ (integers increased by infinity, denoted $\omega$). These functions are studied modulo an equivalence relation $\approx$ which eliminates any information regarding the exact values of the functions, but sufficiently preserves information to test properties such as "is the function bounded?".

This theory is the continuation of work whose objective was the resolution precise decision problems in automata theory and logic. Let's quote:

**The star height.** Input: a regular language $L$ and an integer $k$. Exit: yes if the language $L$ can be described by means of a rational expression of height $k$, no otherwise. (problem posed by Eggan in 63 [8]).

**The finite power.** Input: a regular language $L$. Output: yes if it exists $n$ such that $L^n = L^*$, no otherwise (problem due to Brzozowski).

**The substitution finished.** Input: two languages $K, L$. Output: yes if there is a substitution $\sigma$ sending each letter on a finite language, and such that $\sigma(L) = K$, no otherwise.

**The bound on fixed-point.** In this problem, which would require more formalism to be introduced formally, it is a question of deciding whether the fixed point of a logical formula can be reached in a bounded number of iterations on a class of structure. This problem, studied in model theory, comes in several modalities depending on the choice of logic and the class of structure. Here, this is second-order monadic logic, on the class of words.

All these questions have been resolved [18,7,11,10,12,13,15,19,9,1,14,2] using a reduction to questions of the form "decide if the function $F$ is bounded", where $F$ is a finitely described function (e.g. using automata or regular expressions having counting capabilities), and calculated based on the inputs of the problem. This course aims to describe a theory, parallel to the theory of regular languages, and providing systematic tools for resolving such boundary existence questions.

## Notations

We use the following notations: $\mathbb{A}$ is a (finite) alphabet, $\mathbb{A}^*$ is the set of words on this alphabet, $\mathbb{A}^+$ is the set words of at least one letter. $\varepsilon$ denotes the stop word. For $a$ a fixed letter, and $u$ a word, $|u|_a$ represents the number of occurrences of the letter $a$ in the word $u$.

The set $\omega+1$ is $\{0, 1, \dots\} \cup \{\omega\}$, i.e., the augmented integers of a "largest infinite element" $\omega$. All maximum computations and minimum are carried out in this set. We therefore take as conventions that $\min \emptyset = \omega$ and $\max \emptyset = 0$.

## 1 Definition of B-automata

This first section aims to introduce the B-automata model. B-automata are finite automata which associate a value in $\omega+1$ with each word. They use counters that they can increment, reset, and observe. The function they calculate is the minimum over all runs of the largest value of a counter observed.

### 1.1 Cost of a sequence

We begin the description of automata by describing more precisely how calculate the result of a series of *actions* on a *counter*. We use three *elementary actions*: increment ($\mathtt{i}$=‘*increment*’), reset to 0 ($\mathtt{r}$=‘*reset*’), and test ($\mathtt{c}$=‘*check*’). Consider a sequence $u \in \{\mathtt{i}, \mathtt{c}, \mathtt{r}\}^*$, and a counter that can take integer values. The sequence is performed from left to right. At the beginning the counter has the value 0, then it evolves with each letter encountered, in the following way:

– if the letter is $\mathtt{i}$, the counter value is increased by 1,

– if the letter is $\mathtt{r}$, the counter value is reset to 0,
– if the letter is $\mathtt{c}$, the counter value does not change, but is *observed*.

We define $C(u)$ as the set of observed values, and we define the *cost* of the sequence as $cost_B(u) = \max C(u)$ (recall that $\max \emptyset = 0$ by convention).

A more direct way to calculate the cost of a sequence is for example to use the next fact.

**Fact 1** $cost_B(u) = \max\{|v|_i \ : \ u = u'v\mathtt{c}u'',\ |v|_{\mathtt{r}} = 0\}$ .

Let $\Gamma$ be a finite set of *counters*, all *actions (simple)* on $\Gamma$ is $\mathrm{A}_\Gamma = \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$. Given a word $u$ on the alphabet $\mathrm{A}_\Gamma$, we define $C(u)$ as the union of $C(u^\gamma)$ where $u^\gamma$ is the word obtained by projecting each letter of $u$ on its component $\gamma \in \Gamma$. We define $cost_B(u)$ as $\max C(u)$.

This is obtained in the same way using the next fact.

**Fact 2** $cost_B(u) = \max\{cost_B(u^\gamma) \ : \ \gamma \in \Gamma\}$ .

## 1.2 B-automata

A *B-automaton* $\mathcal{A} = \langle Q, \mathbb{A}, I, F, \Gamma, \Delta\rangle$ is described by:

– a finite set of *states* $Q$,
– a finished *alphabet* $\mathbb{A}$,
– a set of *initial states* $I \subseteq Q$,
– a set of *final states* $F \subseteq Q$,
– a set of *transitions* $\Delta \subseteq Q \times \mathbb{A} \times \mathrm{A}_\Gamma \times Q$.

A *run* $\sigma$ of $\mathcal{A}$ on $u \in \mathbb{A}^*$ is a sequence $\sigma = (p_0, a_1, t_1, p_1) \dots (p_{n-1}, a_n, t_n, p_n)$ such that:

– $(p_{i-1}, a_i, t_i, p_i) \in \Delta$,
– $a_1 \dots a_n = u$,
– $p_0 \in I$,
– $p_n \in F$.

Its cost is $cost_B(\sigma) = cost_B(t_1 \dots t_n)$.

We define the value of a word $u \in \mathbb{A}^*$ for $\mathcal{A}$ as:

$$[\![\mathcal{A}]\!]_B(u) = \min\{cost_B(\sigma) \ : \ \sigma \text{ run of } \mathcal{A} \text{ on } u\} \qquad \text{(calculated in } \omega + 1)$$

(recall that $\min \emptyset = \omega$, by convention)

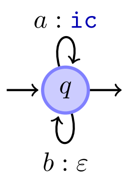
*Remark 1.* If there is no run on $u$, then $[\![\mathcal{A}]\!]_B = \omega$.

*Remark 2.* If $\mathcal{A}$ has no counter (i.e., $\Gamma = \emptyset$) then $\mathcal{A}$ can be seen as a non-deterministic automaton accepting the regular language $L = \mathcal{L}(\mathcal{A})$. We then have for every word $u$:

$$[\![\mathcal{A}]\!]_B(u) = \chi_L(u) = \begin{cases} 0 & \text{si } u \in L, \\ \omega & \text{otherwise.} \end{cases}$$

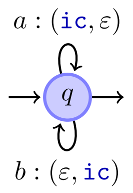
In particular, if $\mathcal{A}$ and $\mathcal{A}'$ are B-automata without counters, then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ if and only if $[\![\mathcal{A}]\!] \geqslant [\![\mathcal{A}']\!]_B$.

*Example 1.* The following B-automaton counts the number of occurrences of the letter $a$.



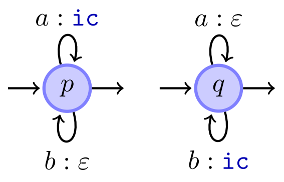
We use the following conventions. States are represented by circles. Initial states have an incoming arrow with no origin, and final states are origins of an arrow without destination. A transition $(p, a, t, q)$ is represented as an arrow of origin $p$, destination $q$, labeled by $a : t$. In this example, as there is only one counter, $t$ is represented as the action on the single counter.

*Example 2.* The following B-automaton computes the maximum of the number of occurrences of letter $a$ and the number of occurrences of letter $b$. It makes use of two counters:


$a : (\mathtt{ic}, \varepsilon)$
$q$
$b : (\varepsilon, \mathtt{ic})$

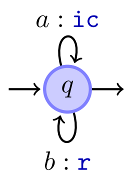
This time, we use two counters, and the actions are therefore represented as a pair consisting of the action on the first counter, followed by the action on the second counter.

*Example 3.* The following one-counter B-automaton calculates the minimum of the number of occurrences of the letter $a$ and the number of occurrences of the letter $b$.

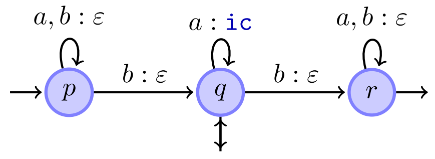


This time, the automaton must guess, using non-determinism, whether it counts the number of occurrences of $a$ or the number of occurrences of $b$.

*Example 4.* The following B-automaton calculates the maximum length of a segment formed only by $a$.



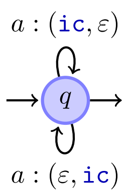
*Example 5.* The following B-automaton calculates the minimum length of a segment formed only by $a$ (a segment being delimited here, by, on the left, either the beginning of the word or a letter $b$, and on the right, either the end of the word or a letter $b$).

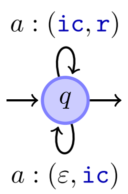


We use the abbreviation $a, b : t$ annotating an arrow of origin $p$ and destination $q$, to denote that there exist the two transitions $(p, a, t, q)$ and $(p, b, t, q)$.

This automaton works by guessing, using its non-determinism, the segment of $a$ that is to be measured.

*Exercise 1.* Determine what the following two-counter B-automata calculates:


$a : (\mathtt{ic}, \varepsilon)$
$q$
$a : (\varepsilon, \mathtt{ic})$


$a : (\mathtt{ic}, \mathtt{r})$
$q$
$a : (\varepsilon, \mathtt{ic})$

*Exercise 2.* Show that it is impossible to determinize B-automata. Idea: take the second automaton of the previous example, and notice that deterministic automata on a letter have a "quasi linear" behavior, i.e.: if $[\![\mathcal{A}]\!]_B(a^n)$ tends to infinity, then $[\![\mathcal{A}]\!]_B(a^{kn}) \geqslant n$ for a certain integer $k$.

### 1.3 Elementary results

In this section, we establish some closure results on functions computed by B-automata.

**Proposition 1.** *The support of a function defined by B-automata is regular (i.e., the set of finite-valued words).*

Proof idea: remove actions on counters to obtain a non-deterministic automaton accepting a language.

**Proposition 2.** *The set of words of value $n$ is regular.*

Idea: it is possible, by means of a finite automaton, to have counters "up to $n$".

A finite automaton can therefore simulate a B-automaton for all values of the counters less than $n$.

**Proposition 3.** *The functions computed by B-automaton are closed under* min*.*

Idea: perform the disjoint union of the two automata. As an example, we can observe the relationship between the automaton of example 1 and the automaton of example 3.

**Proposition 4.** *The functions computed by B-automaton are closed under* max*.*

Idea: this time we are going to build a product automaton working on the disjoint union of counters. As an example, we can observe the relationship between the automaton of the example 1 and the automaton of the example 2.

**Proposition 5.** *The functions calculated by B-automaton are closed under* min*-projection (let $f$ be a letter-to-letter morphism, and $\mathcal{A}$ be a B-automaton, build $\mathcal{A}'$ such that $\mathcal{A}'(u) = \min\{\mathcal{A}(v) \ : \ f(v) = u\}$).*

Idea: replace every transition $(p, a, t, q)$ by a transition $(p, f(a), t, q)$.

## 2 Equivalence "of bounds"

It is possible, as a consequence of a result of Krob [16], to obtain the following undecidability result:

**Theorem 3.** *The problem of equality of functions computed by B-automata is undecidable.*

As mentioned in the introduction, the goal of this theory is not to precisely compute the values produced by B-automata, but rather to decide questions like: "is the function computed by a B-automata bounded?"

For this reason, we allow ourselves to study the functions computed by B-automata modulo an equivalence relation that preserves the properties of the type "existence of bound", but eliminates enough information to circumvent undecidability results like that of Theorem 3. The purpose of this section is to introduce an equivalence relation that allows us to achieve this goal.

## 2.1 Definition

Let $E$ be a set, and $f, g$ two applications of $E$ in $\omega + 1$, then $f$ is *dominated* by $g$, denoted $f \preccurlyeq g$ if for any subset $X \subseteq E$, if $g$ is bounded on $X$ then $f$ is also bounded on $X$. The notation $f \approx g$ means that, at the same time, $f \preccurlyeq g$ and $g \preccurlyeq f$. We call *cost function* (on $E$) an equivalence class for $\approx$.

*Example 6.* $|\cdot|_a \preccurlyeq |\cdot|$, indeed, on any subset of words of bounded length, the number of occurrences of $a$ is also bounded.
On the other hand $|\cdot|_a \not\preccurlyeq |\cdot|_b$, because on the set $X = a^*$, $|\cdot|_b$ is bounded, while $|\cdot|_a$ is not.
We can also note that if $f \leqslant g$ then $f \preccurlyeq g$.

To show that functions are comparable for $\preccurlyeq$ , it is useful to use a refinement of $\preccurlyeq$ . We call *correction function* an increasing function $\alpha$ from $\omega$ to $\omega$ that we extend by $\alpha(\omega) = \omega$. We then set:

$$f \preccurlyeq_\alpha g \qquad \text{if} \qquad f \leqslant \alpha \circ g\ ,$$

and $f \approx_\alpha g$ if $f \preccurlyeq_\alpha g$ and $g \preccurlyeq_\alpha f$. The idea is that $f \preccurlyeq_\alpha g$ means that, once stretched by $\alpha$, $g$ is greater than $f$.

Note that the relation $\preccurlyeq_\alpha$ is not transitive. However, it satisfies the following properties:

**Fact 4** *If $f \preccurlyeq_\alpha g$ then $f \preccurlyeq_{\alpha'} g$ for all $\alpha' \geqslant \alpha$.*
*If $f \preccurlyeq_\alpha g \preccurlyeq_{\alpha'} h$ then $f \preccurlyeq_{\alpha' \circ \alpha} h$.*

*Example 7.* We have $|\cdot|_a + |\cdot|_b \approx_{\times 2} \max(|\cdot|_a, |\cdot|_b)$, in fact:

$$\max(|\cdot|_a, |\cdot|_b) \leqslant |\cdot|_a + |\cdot|_b \leqslant 2\max(|\cdot|_a, |\cdot|_b)\ .$$

**Proposition 6.** *The following properties are equivalent:*

1. $f \preccurlyeq g$,
2. $\forall N. \exists M. \forall x \in E.\ g(x) \leqslant N \to f(x) \leqslant M$.
3. *there exists $\alpha$ such that $f \preccurlyeq_\alpha g$.*

*Remark 3.* For any language $L$, $\chi_L \preccurlyeq \chi_K$ if and only if $K \subseteq L$.

*Example 8.* On $E = N$, $(n \mapsto n) \approx (n \mapsto 2^n) \succ (\frac{n+(-1)^n n}{2}) \succ 0$.
On $E = N^2$, $(\max) \approx_{\times 2} (+)$
On $E = \{\text{set of finite trees}\}$, $\mathit{size} \approx_{n \mapsto n^n} \max(\mathit{degmax}, \mathit{height})$
On $E = \{a, b\}^*$, $|\cdot| = |\cdot|_a + |\cdot|_b \approx \max(|\cdot|_a + |\cdot|_b) \succ |a| \succ 0$.

**Fact 5** *If $f \approx f'$ and $g \approx g'$ then $\min(f, g) \approx \min(f', g')$ and $\max(f, g) \approx \max(f', g')$. If $f_i \approx_\alpha g_i$ for all $i \in I$, then $\inf_{i \in I} f_I \approx_\alpha \inf_{i \in I} g_I$. Same for $\sup$.*

**Fact 6** *The set of functions of $E$ in $\omega + 1$ quotient by $\approx$ is a lattice.*
*It has a smaller class: the class of bounded functions.*
*It has a larger class: the class of the constant function equal to $\omega$.*
*If $E$ is countable, the 'sublattice' induced by the functions of $E$ in $\omega$ has a maximal element (the class of one/all bijections of $E$ onto $\omega$).*

**Proposition 7.** *The set of functions from $E$ infinite countably in $\omega$ has a cardinality of the continuum of equivalence classes.*

*Proof.* We assume, without loss of generality, that $E$ is the set of applications from $\mathbb{N}$ to $\mathbb{N}$ that are zero almost everywhere. There are a countable number of them. For all $I \subseteq \mathbb{N}$ and $g \in E$, we assume $f_I(g) = \max\{g(i) \ :\ i \in I\}$.

Let $I \neq J$. We assume, without loss of generality, that there exists $i \in I \setminus J$. Consider the function $g_n$ that is zero everywhere except for $g_n(i) = n$. We then have $f_I(g_n) = n$, and $f_J(g_n) = 0$. Thus, $f_J$ is bounded on $Y = \{g_n \ :\ n \in \mathbb{N}\}$, while $f_I$ is not. We deduce $f_I \not\approx f_J$.

Since the set of subsets of $\mathbb{N}$ has the cardinality of the continuum, we deduce that there are as many equivalence classes for the relation $\approx$.

## 3 B-automata modulo $\approx$

### 3.1 First remarks

*Example 9.* $|\cdot|_a + |\cdot|_b = \max(|\cdot|_a + |\cdot|_b)$ We have seen that these two functions are accepted by B-automata. These are therefore equivalent modulo $\approx$.

The following proposition shows that, even considered modulo $\approx$, B-automata cannot be deterministic.

**Proposition 8.** *The function $a^n b^m \mapsto \min(m, n)$ cannot be recognized by a deterministic B-automaton modulo $\approx$.*

*Proof.* By the contrapositive, suppose that there exists a correction function $\alpha$ and a B-automaton $\mathcal{A}$ that defines a function $\approx_\alpha$ -equivalent to $a^n b^m \mapsto \min(m, n)$. Let us denote by $\delta(u)$ the state reached from the initial state by reading the word $u$ as input.

We begin by analyzing the structure of the automaton. Let us first note that the word $a^m b^n$ does not have the value $\omega$, and thus $\delta(a^m b^n)$ exists for all $m, n$, and is final. Since the set of states is finite, there exist $k, l$ such that $\delta(a^k) = \delta(a^{k+l})$.

Similarly, there exist $k', l'$, such that $\delta(a^k b^{k'}) = \delta(a^k b^{k'+l'})$.

Suppose there exists a counter $\gamma$ such that on the loop traversed while reading $a^l$ from state $\delta(a^k)$, $\gamma$ is incremented, but not reset to 0. So, we deduce that $[\![\mathcal{A}]\!](a^{k+nl}) \geqslant n$, and therefore $[\![\mathcal{A}]\!]$ is not bounded on $a^{k+nl}$, while $a^n b^m \mapsto \min(m, n)$ is. Contradiction. Similarly, it is impossible for a counter to be incremented but not reset to 0 when traversing the original loop $\delta(a^k b^{k'})$ obtained by reading the word $b^{l'}$. Thus, we know that all counters are either reset to 0 or never incremented in these loops. We deduce that $[\![\mathcal{A}]\!](a^{k+nl} b^{k'+nl'}) \leqslant k + l + k' + l'$, and therefore $[\![\mathcal{A}]\!]$ is bounded on $\{a^{k+nl} b^{k'+nl'} \ : \ n \in N\}$, while $a^n b^m \mapsto \min(m, n)$ is not. We again arrive at a contradiction.

### 3.2 Equivalence with B-automata with $\varepsilon$-transitions

A *B-automaton with $\varepsilon$-transitions* is a B-automaton over the alphabet $A \cup \{\lambda\}$. Given $u \in A \cup \{\lambda\}$, $\overline{u}^\lambda$ is the word of $A^*$ obtained from $u$ by eliminating all occurrences of the letter $\lambda$. We set for all $u \in A^*$:

$$[\![\mathcal{A}]\!]_B^\lambda(u) = \min\{[\![\mathcal{A}]\!]_B(v) \ : \ \overline{v}^\lambda = u\} \ .$$

We note that in the case without counters, this definition is consistent with the usual notion of automata with $\varepsilon$-transitions.

The objective is to establish the following proposition.

**Proposition 9.** *B-automata with $\varepsilon$-transitions are $\approx$-equivalent to B-automata (without $\varepsilon$-transitions).*

We define the product operation $\cdot$ on $\varepsilon, \texttt{ic}, \texttt{r}$ as the maximum for the order $\varepsilon < \texttt{ic} < \texttt{r}$. This product is extended component by component to $\mathrm{A}_\Gamma = \{\varepsilon, \texttt{ic}, \texttt{r}\}^\Gamma$. Let $\pi : \{\varepsilon, \texttt{ic}, \texttt{r}\}^* \to \{\varepsilon, \texttt{ic}, \texttt{r}\}$ be the extension of the product to a word of any length $u \in \mathrm{A}_\Gamma{}^*$ (with $\pi(\varepsilon) = \varepsilon$).

**Lemma 1.** *For all $u_1, \ldots, u_n \in \mathrm{A}_\Gamma{}^*$, $cost_B(\pi(u_1) \ldots \pi(u_n)) \leqslant cost_B(u)$, where $u = u_1 \ldots u_n$. If $|u_i| \leqslant k$ for all $i = 1, \ldots, n$, then $cost_B(u) \leqslant \alpha(cost_B(\pi(u_1) \ldots \pi(u_n)))$ where $\alpha(x) = k(x + 2)$.*

*Proof.* Let $N = cost_B(\pi(u_1) \ldots \pi(u_n))$. By definition of $cost_B$, there exists a factor $u_i \ldots u_j$ of $u$ and a counter $\gamma \in \Gamma$ such that $\gamma$ is never reset to 0 in $\pi(u_i) \ldots \pi(u_j)$ and is incremented $N$ times in $\pi(u_i) \ldots \pi(u_j)$. From the definition of the product, we deduce that $\gamma$ is never reset to 0 in $u_i \ldots u_j$, and that there exist $N$ indices $l \in \{i, \ldots, j\}$ such that $\gamma$ is incremented in $u_l$. We deduce that $cost_B(u_i \ldots u_j) \geqslant N$, and consequently $cost_B(u) \geqslant N$.

Second implication. Let $N = cost_B(u)$. There exists a counter $\gamma$ and a factor $v$ of $u$ such that $\gamma$ is never reset to 0 in $v$, and is incremented $N$ times. If $v$ is the factor of one of the $u_i$, then it is of length at most $k$, and we obtain directly $cost_B(u) = N \leqslant |v| \leqslant k \leqslant \alpha(cost_B(\pi(u_1) \ldots \pi(u_n)))$. Otherwise, $v$

decomposes into $v'u_i \dots u_j v''$, where $v'$ is a suffix of $u_{i-1}$ (or empty), and $v''$ is a prefix of $u_{j+1}$ (or empty). Since $|v'| \leqslant |u_{i-1}| \leqslant k$, and $|v''| \leqslant |u_{j+1}| \leqslant k$, we deduce that $\gamma$ is incremented at least $N - 2k$ times in $u_i \dots u_j$. Let $I$ be the set of $l = 1 \dots j$ such that $\pi(u_l) = \texttt{ic}$. Any increment of $u_i \dots u_j$ occurs in a $u_l$ for $l \in I$, moreover each $u_l$ for $l \in I$ contains at most $k$ increments of $\gamma$. We deduce that $N - 2k \leqslant k|I|$, and consequently $cost_B(u) \leqslant k(|I| + 2) \leqslant \alpha(cost_B(\pi(u_1) \dots \pi(u_n)))$ with $\alpha(x) = k(x + 2)$. □

**Construction:** We start from the automaton $\mathcal{A} = \langle Q, A, I, F, \Gamma, \Delta \rangle$ with $\varepsilon$-transitions. Let us now consider $\sigma = (p_0, a_1, t_1, p_1) \dots (p_{n-1}, a_n, t_n, p_n)$ a path in $\mathcal{A}$, let us denote by $\pi_1(\sigma)$ the word $a_1 \dots a_n$, and $\pi_2(\sigma)$ the value $\pi(t_1 \dots t_n)$. In this case $\sigma$ is called $a_1 \dots a_n$-path.

We construct the automaton $\mathcal{A}' = \langle Q, A, I, F, \Gamma, \Delta' \rangle$ where $\Delta'$ contains a transition $(p, a, t, q)$ if and only if there exists a path $\sigma'$ from $p$ to $q$ in $\mathcal{A}'$ such that $\pi_1(\sigma') = a$ and $\pi_2(\sigma') = t$ (the run $\sigma'$ is called a *witness* of $(p, a, t, q)$). If $\mathcal{A}(\varepsilon) < \omega$, we add to $\mathcal{A}'$ an isolated state that is both initial and final.

**Lemma 2.** *If there exists a $\varepsilon$-path $\sigma$ from $p$ to $q$, then there exists a $\varepsilon$-path $\sigma'$ from $p$ to $q$ of length at most $|\mathrm{A}_\Gamma||Q| - 1$ such that $\pi_2(\sigma) = \pi_2(\sigma')$.*

*Any transition $(p, a, t, q)$ from $\mathcal{A}'$ admits a witness path of length at most $K = 2|\mathrm{A}_\Gamma||Q| - 1$.*

*Proof.* It suffices to show that for all $\varepsilon$-paths $\sigma$ from $p$ to $q$ of length at least $|\mathrm{A}_\Gamma||Q|$ there exists a shorter $\varepsilon$-path $\sigma'$ from $p$ to $q$ such that $\pi_2(\sigma) = \pi_2(\sigma')$. having the same initial and final states, and such that $\pi_1(\sigma') = \varepsilon$ and $\pi_2(\sigma') = \pi_2(\sigma)$. To do this, we just need to note that $\sigma$ can be decomposed into $\sigma_1\sigma_2\sigma_3$ such that $|\sigma_2| \geqslant 1$, $\sigma_1$ and $\sigma_2$ have the same final state and $\pi_2(\sigma_1) = \pi_2(\sigma_1\sigma_2)$ (to do this, we note that the pair $(\pi_2(\sigma_1), q)$ can only take $|\mathrm{A}_\Gamma||Q|$ values, so after $|\mathrm{A}_\Gamma||Q|$, there is at least one repetition). We deduce that $\sigma' = \sigma_1\sigma_3$ is suitable. Applying this shortening argument by induction, we can reduce the length of $\sigma$ to at most $|\mathrm{A}_\Gamma||Q| - 1$.

Consider now a witness path $\sigma$ of $(p, a, t, q)$, it can be written as $\sigma_1\delta\sigma_2$ where $\sigma_1$ and $\sigma_2$ are $\varepsilon$-paths and $\pi_1(\delta) = a$. By the previous argument we construct $\sigma'_1$ from $\sigma_1$ and $\sigma'_2$ from $\sigma_2$. Let $\sigma' = \sigma'_1\delta\sigma'_2$. By construction $\sigma'$ is a witness to $(p, a, t, q)$, and the length of $\sigma'$ is at most $2(|\mathrm{A}_\Gamma||Q| - 1) + 1$. □

It is now possible to complete the proof of the Proposition 9 by establishing that for all $u \in A^*$, $[\![\mathcal{A}']\!]_B(u) \leqslant \mathcal{A}(u)$, and $[\![\mathcal{A}]\!]_B(u) \leqslant K([\![\mathcal{A}']\!]_B(u) + 1)$ (which implies, $[\![\mathcal{A}']\!]_B \approx [\![\mathcal{A}]\!]_B$).

*Proof.* First inequality. If $[\![\mathcal{A}]\!](u) = \omega$, we directly obtain $[\![\mathcal{A}']\!]_B(u) \leqslant [\![\mathcal{A}]\!](u)$. Otherwise, we separate two cases. If $u = \varepsilon$. Since $[\![\mathcal{A}]\!](\varepsilon) < \omega$, there exists (by construction) an initial and final state in $\mathcal{A}'$, and consequently $[\![\mathcal{A}']\!]_B(\varepsilon) = 0$. We deduce $[\![\mathcal{A}']\!]_B(\varepsilon) \leqslant [\![\mathcal{A}']\!]_B(\varepsilon)$. Otherwise, $u = a_1 \dots a_n$ is nonempty. Since $[\![\mathcal{A}]\!](u) < \omega$, there exists an run $\sigma$ of $[\![\mathcal{A}]\!]$ with cost $[\![\mathcal{A}]\!](u)$ on the word $u$. This run decomposes into $\sigma_1 \dots \sigma_n$, where $\pi_1(\sigma_i) = a_i$. Let $p_0$ be the initial state of $\sigma_1$ and $p_i$ be the final state of $\sigma_i$. The run $\sigma_i$ is a witness of $\delta_i = (p_{i-1}, a_i, \pi_2(\sigma_i), p_i)$ for all $i = 1, \dots, n$. We deduce that $\sigma' = \delta_1 \dots \delta_n$ is an run of $\mathcal{A}'$ on $u$. Thanks to Lemma 1, its cost is at most $cost_B(\sigma) = [\![\mathcal{A}]\!](u)$.

Conversely. Let $u$ be a word. If $u = \varepsilon$ and $[\![\mathcal{A}']\!]_B(u) < \omega$, then necessarily $\mathcal{A}'$ contains a state that is both initial and final. Either this state was already present in $\mathcal{A}$, in which case $[\![\mathcal{A}]\!]_B(u) = 0$. Or this state is the state that we added because $[\![\mathcal{A}]\!](\varepsilon) < \omega$. By Lemma 2, this means that $\varepsilon$ is accepted by a one run of length at most $|\mathrm{A}_\Gamma||Q| - 1$. We deduce $[\![\mathcal{A}]\!](u) \leqslant K \leqslant K([\![\mathcal{A}']\!]_B(u) + 1)$. Otherwise $u \neq \varepsilon$. Let $\sigma'$ be an run of $\mathcal{A}'$ on $u = a_1 \dots a_n$. This run is written as $\sigma' = \delta'_1 \dots \delta'_n$. Thanks to the **??** there exists for all $i = 1, \dots, n$ a path $\sigma_i$ witness of $\delta'_i$ of length at most $K$. Consider the path $\sigma = \sigma_1 \dots \sigma_n$. By construction, this is an run of $\mathcal{A}$ on $u$. Moreover, by Lemma 1, $cost_B(\sigma) \leqslant K\,cost_B(\sigma')$. □

### 3.3 Equivalence with hierarchical B-automata

**Definition of hierarchical B-automata** We restrict the possible use of counters. The counters are now totally ordered (assumed to be $1, \dots, k$), and incrementing or resetting a counter resets all lower counters.

**Definition 1.** *A B-automaton is said to be hierarchical if its set of counters is of the form $\Gamma = \{1, \dots, k\}$, and the only actions $\mathrm{H}_k$ on the counters are:*

– $\mathtt{R}_l$ *for* $l = 0, \ldots, k$, *such that* $\mathtt{R}_l(\gamma) = \begin{cases} \mathtt{r} & \text{if } \gamma \leqslant l \\ \varepsilon & \text{otherwise} \end{cases}$

– $\mathtt{I}_l$ *for* $l = 1, \ldots, k$, *such that* $\mathtt{I}_l(\gamma) = \begin{cases} \mathtt{r} & \text{if } \gamma < l \\ \mathtt{ic} & \text{if } \gamma = l \\ \varepsilon & \text{otherwise.} \end{cases}$

*Remark 4.* We can use $\mathtt{R}_0$ even if there is no counter of this number. We can also simply write this action as $\varepsilon$.

*Remark 5.* For the product used above, we obtain that $c \cdot c'$ for $c, c'$ in $\mathrm{H}_k$ is the maximum in order:

$$\mathtt{R}_0 < \mathtt{I}_1 < \mathtt{R}_2 < \quad \ldots \quad < \mathtt{I}_k < \mathtt{R}_k \ .$$

We deduce using the same construction that $\varepsilon$-transitions can be eliminated from hierarchical B-automata.

**Equivalence**

**Theorem 7.** *Hierarchical B-automata are $\approx$-equivalent to B-automata.*

We use a technique from the theory of automata on words and infinite trees.

**Step 1:**We construct a hierarchical (deterministic) automaton $LAR_\Gamma$ on the alphabet $\mathrm{A}_\Gamma$ such that:

$$[\![LAR_\Gamma]\!]_B \approx cost_B \ .$$

**Construction:** We set $k = |\Gamma|$. $LAR_\Gamma$ is defined by:

– the states are permutations of $\Gamma$, i.e., bijections of $\{1, \ldots, k\}$ onto $\Gamma$,
– the input alphabet is $\mathrm{A}_\Gamma$,
– the counters are $\{1, \ldots, k\}$,
– from the state $\pi$, and reading the letter $t \in \mathrm{A}_\Gamma = \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^\Gamma$, we compute:

$$i = \max\{l \ : \ t(\pi(l)) = \mathtt{ic}\} \qquad \text{and} \qquad r = \max\{l \ : \ t(\pi(l)) = \mathtt{r}\} \qquad \text{(0 by default).}$$

we use the transition:
  - $(\pi, t, \mathtt{I}_i, \pi')$ if $i > r$,
  - $(\pi, t, \mathtt{R}_r, \pi')$ if $r \geqslant i$ (and as a special case $\mathtt{R}_0 = \varepsilon$ if $i = r = 0$).

where $\pi'$ is obtained from $\pi$ by putting the counters $\gamma$ such that $t(\gamma) = r$ at the head of the permutation, and leaving the other relative orders unchanged.

**Lemma 3.** *Consider a transition $(\pi, t, h, \pi')$ from $LAR_\Gamma$. Consider the indices in $\pi$ and $\pi'$ of a counter $\gamma$, then:*

– *if $\gamma$ is not reset to zero in $h$, its index can only increase,*
– *if $\gamma$ is not reset to zero in $h$ then its index strictly increases if and only if a counter with a higher index is reset to 0.*

*Proof.* By construction. It suffices to show for a letter, then by induction on the length.

**Lemma 4.** *If $u$ is a sequence of hierarchical increments of length $n^k$, then $cost_B(u) \geqslant n$.*

*Proof.* By induction on $k$ we prove that any sequence of hierarchical increments touching only counters less than or equal to $k$ and of length at least $n^k$ has a cost at least equal to $n$. For $k = 0$, the length of the sequence is zero, and therefore the property is true. Otherwise, the sequence $u$ is written as $u_0 \mathtt{I}_k u_1 \ldots \mathtt{I}_k u_l$ where the $u_i$'s do not contain any increments of $k$. If one of the $u_i$'s has a size at least equal to $n^{k-1}$, we conclude with the induction hypothesis. Otherwise, $n^k \leqslant |u| = |u_0| + |u_1| + \cdots + |u_l| + l < n^{k-1}(l+1)$. We deduce that $n < l + 1$, i.e., $l \geqslant n$. The cost and of the sequence is therefore at least $n$.

**Lemma 5.** *For all $u \in \mathrm{A}_\Gamma{}^*$, $cost_B(u) \approx_\alpha LAR_\Gamma(u)$ where $\alpha(n) = k(n+1)^k - 1$.*

*Proof.* Let us fix an run $\rho$ of $LAR_\Gamma$ (on a word $u \in \mathrm{A}_\Gamma{}^*$). Let $n = cost_B(\rho)$. This means that there exists a factor $\rho'$ of $\rho$ and $l \in \{1, \ldots, k\}$ such that $\mathtt{I}_l$ is performed $n$ times, and that all the other $\mathtt{I}_m$ and $\mathtt{R}_m$ take place for $m < l$. The counter $\pi(m)$ is unchanged in $\rho'$, so $\pi(m)$ is incremented $n$ times in $\rho'$. Hence $cost_B(u) \geqslant n$. We obtain $LAR_\Gamma(u) \leqslant cost_B(u)$.

Conversely: Suppose that $cost_B(u) \geqslant kn^k$, this means that there is a factor $\rho'$ of $\rho$ and a counter $\gamma$ such that $\gamma$ is incremented $kn^k$ times in $\rho'$, without ever being reset to 0.

According to Lemma 3, the index of $\gamma$ can only increase in $\rho'$. This means that there exists a factor $\rho''$ of $\rho'$ in which $\gamma$ keeps the same index $l$ while being incremented $n^k$ times. During $\rho'$ no $\mathtt{R}_{l'}$ for $l' \geqslant l$ is performed, and at least $n^k$ hierarchical increments $\mathtt{I}_{l'}$ for $l' \geqslant l$ are performed. By **??**, we obtain $cost_B(\rho') \geqslant n$, and consequently $LAR_\Gamma(u) \geqslant n$ . We deduce that:

$$cost_B(u) \preccurlyeq_\alpha LAR_\Gamma(u) \qquad \text{for } \alpha(n) = k(n+1)^k - 1 \ .$$

**Step 2:** We start with a B-automaton $\mathcal{A}$ (counters $\Gamma$) and we perform the product with $LAR_\Gamma$ to obtain an equivalent hierarchical automaton.

**Construction:** $\mathcal{A} = \langle Q, A, I, F, \Gamma, \Delta \rangle$ and $LAR_\Gamma = \langle P, \mathrm{A}_\Gamma, \mathrm{H}_k, \delta \rangle$

We construct $\mathcal{A}' = \langle Q \times P, A, I \times P, F \times P, \mathrm{H}_k, \Delta' \rangle$ where:

$$\Delta' = \{((q,p), a, h, (q',p')) \ : \ (q,a,m,q') \in \Delta, \ (p, \mathbf{M}, h, p') \in \delta\} \ .$$

The hierarchical automaton $\mathcal{A}'$ is $\approx$-equivalent to $\mathcal{A}$.

## 3.4 Equivalence with $B$-regular expressions

A *B-regular expression* (or *B-expression*) respects the following syntax:

$$E ::= \quad \varepsilon \quad | \quad a \quad | \quad E + E \quad | \quad E \cdot E \quad | \quad E^* \quad | \quad E^B \quad (|\emptyset) \ .$$

The semantics $[\![e]\!]_B$ of a regular expression $e$ is an application of $A^*$ in $\omega + 1$ defined by induction:

- $[\![\varepsilon]\!]_B(u) = \begin{cases} 0 & \text{si } u = \varepsilon \\ \omega & \text{otherwise} \end{cases}$,
- $[\![a]\!]_B(u) = \begin{cases} 0 & \text{si } u = a \\ \omega & \text{if not} \end{cases}$,
- $[\![f+g]\!]_B(u) = \min([\![f]\!]_B(u), [\![g]\!]_B(u))$,
- $[\![f \cdot g]\!]_B(u) = \min\limits_{u=vw} \max([\![f]\!]_B(u), [\![g]\!]_B(v))$,
- $[\![f^*]\!]_B(u) = \min\limits_{u=u_1\ldots u_n} \max\limits_{i=1\ldots n} [\![f]\!]_B(u_i)$,
- $[\![f^B]\!]_B(u) = \min\limits_{u=u_1\ldots u_n} \max(n, \max\limits_{i=1\ldots n} [\![f]\!]_B(u_i))$.

*Exercise 3.* Show that for any B-expression $E$ and any $K$, $\{u \ : \ [\![E]\!]_B(u) \leqslant n\}$ is the language obtained by evaluating the rational expression obtained from $E$ by replacing all occurrences of ${}^B$ by ${}^{\leqslant K}$.

**Proposition 10.** *If $e$ is a regular expression describing the language $L$, then*

$$[\![e]\!]_B(u) = \chi_L = \begin{cases} 0 & \text{if } u \in L \\ \omega & \text{otherwise.} \end{cases}$$

*Proof.* By induction on the expression.

*Example 10.* $a^B$, $b^*(ab^*)^B$, $b^*(ab^*)^B + a^*(ba^*)^B$, $(a^*b)^*a^B(ba^*)^*$.

**Theorem 8.** *$B$-regular expressions are $\approx$-equivalent with B-automata.*

*Proof.* (principle) From B-automata to B-expressions. Without loss of generality, we start with a hierarchical B-automaton $\mathcal{A}$. The proof is by induction on the number of transitions of the automaton. If the automaton has no transitions, then two cases are possible: if there exists a state that is both initial and final, the accepted function is $[\![\varepsilon]\!]_B$ otherwise, it is $[\![\emptyset]\!]_B$. If the automaton contains a transition, then we consider the transition $(p, a, h, q)$ whose action $h$ is maximal for the order $\mathtt{R}_0 < \mathtt{I}_1 < \mathtt{R}_1 < \dots$. Let $\mathcal{A}'_{I,F}$, $\mathcal{A}'_{I,p}$, $\mathcal{A}'_{q,p}$ and $\mathcal{A}'_{q,F}$ be the automata obtained from $\mathcal{A}$ by removing the transition $(p, a, h, q)$, and choosing $I$ or $\{q\}$ as initial states, and $F$ or $\{p\}$ as final states in accordance with the index of the automaton. By induction hypothesis, we obtain the corresponding B-expressions $E'_{I,F}$, $E'_{I,p}$, $E'_{q,p}$ and $E'_{q,F}$. We set:

$$E = \begin{cases} E'_{I,F} + E'_{I,p}a(E'_{q,p}a)^B E'_{q,F} & \text{if } h \text{ is of the form } \mathtt{I}_l \\ E'_{I,F} + E'_{I,p}a(E'_{q,p}a)^* E'_{q,F} & \text{otherwise} \end{cases}$$

We show without difficulty that $[\![E]\!]_B \approx [\![\mathcal{A}]\!]_B$.

From B-expressions to B-automata. The proof is by induction on the B-expression. The construction is particularly simple if we allow $\varepsilon$-transitions.

The most interesting case is when the expression is of the form $E^B$. We apply the recurrence hypothesis on an automaton $\mathcal{A}$ which computes an equivalence function to $[\![E]\!]_B$. We construct a new automaton $\mathcal{A}'$ from $\mathcal{A}$ by:

– adding an isolated initial and final state,
– adding a new counter $\gamma$ (we can improve the construction by choosing an already existing counter $\gamma$, provided that it has never been reset to 0 in $\mathcal{A}$),
– adding for any initial state $q$ and any final state $p$ a transition $(p, \lambda, \mathtt{I}_\gamma, q)$, where $\mathtt{I}_\gamma$ sets to 0 all counters, except $\gamma$ to which the action $\mathtt{ic}$ is assigned.

In the case of an expression $E^*$ the same construction is used, with the difference that no counter is added, and that the added transitions reset all the counters to zero. We can easily convince ourselves that this construction transforms a *B-expression* into a B-automaton (in fact, hierarchical) calculating the same cost function.

## 3.5 Decidability of divergence

A function $f$ from $E$ to $\omega + 1$ is *divergent* if for all $n$, the set $f^{-1}(n)$ is finite. In particular, if $E = \omega$, then $f$ is divergent if and only if $\lim f = \omega$.

**Fact 9** *If $f \approx g$, then $f$ is divergent if and only if $g$ is divergent.*

**Theorem 10.** *The problem of the divergence of functions computed by B-automata is decidable.*

*Proof.* Using the Theorem 9, we choose without loss of generality a $B$-regular expression. We study the following properties:

– $\varepsilon$ if its support is $\{\varepsilon\}$, i.e., if it is equivalent to $[\![\varepsilon]\!]_B$,
– NO if its support contains a non-empty word,
– ND if $f$ is non-divergent.

Then we have the following properties:

– if $e = \varepsilon$, then $e$ is $\varepsilon$,
– if $e = a$, then $e$ is NO,
– if $e = f + g$, then $e$ is the max property of f and g for the order $\varepsilon$<NO<ND,
– if $e = f \cdot g$ then $e$ ... make a table.
– if $e = f^*$ then make a table
– if $e = f^B$ then make a table.

□

# 4 Stabilization monoids

## 4.1 Definition

A *semigroup* $\langle S, \cdot \rangle$ is a set $S$ equipped with an associative product operation ($a \cdot (b \cdot c) = (a \cdot b) \cdot c$). If it contains a neutral element 1, i.e., such that $1 \cdot x = x \cdot 1 = x$ for all $x$, it is called a monoid. An *ordered semigroup* $\langle S, \cdot, \leqslant \rangle$ is a semigroup $S$ equipped with a compatible order $\leqslant$, i.e., such that $a \leqslant b$ and $a' \leqslant b'$ implies $a \cdot a' \leqslant b \cdot b'$. An *idempotent* of a semigroup is an element $e$ such that $e \cdot e = e$. The set of idempotents of a semigroup $\mathbf{S}$ is denoted by $E(\mathbf{S})$.

**Definition 2.** *A* stabilization semigroup $\langle S, \cdot, \leqslant, \sharp \rangle$ *is a* finite *ordered semigroup* $\langle S, \cdot, \leqslant \rangle$ *equipped with an operator* $\sharp \colon E(\mathbf{S}) \to E(\mathbf{S})$ *(called the* stabilization*) such that:*

- *for all* $e \leqslant f$ *in* $E(\mathbf{S})$, $e^\sharp \leqslant f^\sharp$*;*
- *for all* $a, b \in S$ *such that* $a \cdot b \in E(\mathbf{S})$ *and* $b \cdot a \in E(\mathbf{S})$*, we have* $(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b$*;*
- *for all* $e \in E(\mathbf{S})$, $e^\sharp \leqslant e$*;*
- *for all* $e \in E(\mathbf{S})$, $(e^\sharp)^\sharp = e^\sharp$*.*

*A* stabilization monoid *is a stabilization semigroup whose underlying semigroup is a monoid, and such that* $1^\sharp = 1$.

*Example 11 ("count" occurrences of the letter a).* We seek to construct a stabilization monoid informally, with the objective of describing the cost function of the function calculating the number of occurrences of the letter $a$, on the alphabet $\{a, b\}$. In informal terms, we seek to construct a stabilization monoid that separates words containing "many" occurrences of the letter $a$, from those that contain only "few".

To achieve this objective, we must separate 3 "types" of words:

- words without occurrences of $a$; let's call this type '$b$' (the letter $b$ is of this type).
- words containing at least one occurrence of $a$, but only a small number of such occurrences; let us call '$a$' the corresponding type (indeed, the word $a$ is of this type),
- words containing a 'large number' of occurrences of $a$'s'; let us call this type '0'.

Our goal is to separate words of type 0, from words of type $a$ or $b$. We therefore set $M = \{a, b, 0\}$.

Of course, repeating a few or a lot of words that do not contain any $a$ can only produce words that do not contain any $a$. This phenomenon is obtained by setting $b \cdot b = b$ and $b^\sharp = b$.

Iterating a small number of words containing a small number of $a$ (at least 1) produces a word of the same nature, thus, we set $a \cdot a = a$. Similarly, concatenating such words with words without $a$, on the left as well as on the right, does not change the number of $a$. We therefore set $a \cdot b = b \cdot a = a$. On the other hand, iterating a large number of words containing at least one occurrence of the letter $a$ produces a word that contains a large number of $a$. We therefore set $a^\sharp = 0^\sharp = 0$.

Using similar informal arguments, we also produce the rules $a \cdot 0 = 0 \cdot a = 0 \cdot 0 = 0$. This is summarized by means of the following table.

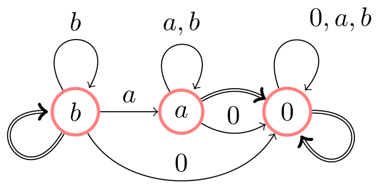
| | $b$ | $a$ | 0 | $\sharp$ |
|---|---|---|---|---|
| $b$ | $b$ | $a$ | 0 | $b$ |
| $a$ | $a$ | $a$ | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

The left part of the table represents the product, the additional column gives the stabilization table (note that in this notation, this column may be only partially defined, since stabilization is defined only on idempotents.).

To complete this definition, it remains to describe the order relation. By definition of a stabilization monoid, it is necessary to set $0 = a^\sharp \leqslant a$. We can verify that this relation (augmented by $x \leqslant x$ for all $x$) does indeed result in the definition of a stabilization monoid.

The idea that we must have of this order is that it specifies how the value of a word can evolve when the cursor separating "a little" from "a lot" varies. For example, the word $a^{100}$ must be considered small if this cursor is set to 1000, and therefore must have the value $a$. On the other hand, if we set the cursor to 10, this same word must be considered as having many occurrences of the letter $a$, and therefore must have the value 0. Thus, there is a sort of "continuum" between the value $a$ and the value 0, since the same word can pass from one to the other when the cursor varies.

It is also nice to represent the stabilization monoid by means of its Cayley graph:



Each vertex represents one of the elements of the monoid, and an arc labeled $y$ connects each vertex $x$ to the vertex $x \cdot y$. A double arrow connects each idempotent $x$ to its "stabilized" $x^\sharp$.

*Remark 6.* A monoid in the usual sense $\langle M, \cdot \rangle$ naturally transforms into a stabilization monoid $\langle M, \cdot, \sharp, \leqslant \rangle$, in which the stabilization of any idempotent $e$ is $e^\sharp = e$, and the order is reduced to the identity. It is easily verified that this definition satisfies all the constraints of a stabilization monoid.

*Exercise 4.* Construct stabilization monoids that "measure"

– the size of the largest segment of consecutive $a$,
– the size of the smallest segment of consecutive $a$,
– on $\{a, b, c\}$, the minimum of the number of $a$ and the number of $b$,
– the number of occurrences of the factor $ab$ in a word.

## 4.2 Towards a semantics of stabilization monoids: $n$-computations

We fix a stabilization monoid $\langle M, \cdot, \sharp, \leqslant \rangle$.

Let $u \in M^*$. We seek to calculate the value of $u$ in the stabilization monoid, for a fixed value of $n$, where $n$ determines what is the limit between "few" and "many". To do this, we need to formalize the notion of computation: a computation is a tree describing the successive steps leading to a result.

**Definition 3.** *An $n$-computation over the word $u \in M^+$ is an unranked*[1] *tree, ordered*[2]*, for which each node $x$ is labeled by an element $v(x) \in M$ called its value and such that:*

– *the leaf labels read from left to right form the word $u$,*
– *for each node $x$, with children $y_1, \ldots, y_k$ (from left to right), one of these situations is true:*
  - *$k = 0$ (leaf),*
  - *$k = 2$, and $v(x) = v(y_1) \cdot v(y_2)$, (product node),*
  - *$2 < k \leqslant n$, and $v(x) = v(y_1) = \cdots = v(y_k) \in E(M)$, (idempotent node),*
  - *$k > n$, $v(y_1) = \cdots = v(y_k) = e \in E(M)$ and $v(x) = e^\sharp$ (stabilization node).*

*The value of a computation tree is the value of its root.*

In the proofs, we will denote $[a]$ the computation restricted to a leaf of value $a$. We will denote by $[a](t_1 \cdot t_2)$ the computation of value $a$ formed by a product node, and whose child subtrees are computations $t_1$ and $t_2$. We will also use the similar notation $[a](t_1 \ldots t_k)$ to denote idempotent and stabilization nodes. These notations will also be used for the variant notions of under- and over-computations.

Two examples of computations are presented in figure 1. The essential idea is that a computation represents a computation leading to its value as a result. Thus, given a word $u \in M^+$, and $n$ we would like to define the $n$-value of a word $u$ in $M^+$ as the value of one of the $n$-computations on $u$.

We must then solve the following two questions:

[1] There is no limit on the number of children of a node.
[2] There is a total order on the children of a node, informally referred to as the left to right order.

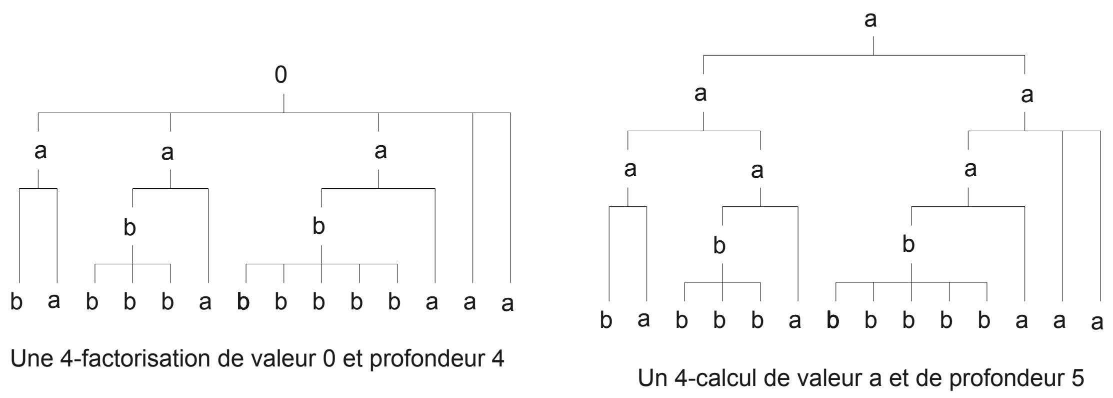

Une 4-factorisation de valeur 0 et profondeur 4
Un 4-calcul de valeur a et de profondeur 5

**Fig. 1.** Two 4-computation trees on the word *babbbabbbbbaaa* in the monoid counting the number of occurrences of $a$.

– Does every word admit an $n$-computation for all $n$?
– Do two $n$-computations of the same word have the same value?

The answer to the first question is a priori elementary: it suffices to use only binary nodes, and to construct a factorization on this principle. This solution is obviously not satisfactory. Indeed, it does not capture the semantics we are looking for since no node using a stabilization is created in this way. This problem comes from the fact that we are trying, at the same time, to control another parameter: the depth of the computation (i.e., its height). We can notice that from the moment the depth is bounded, while the length of the words is not, we find ourselves obliged to construct a computation, to use nodes implying at least one stabilization.

The answer to the first question remains positive, but this time requires the use of more subtle arguments; in this case, we must resort to Green's relations. This is in fact a variation on a result of Simon on monoids, the theorem of forests of factorizations [20], of which there are several other proofs [17,4,3].

**Lemma 6 (admitted).** *For all $u \in M^+$ and all $n$, there exists an $n$-computation of depth at most $3|M|$.*

The answer to the second question is negative, as shown by the two examples of 4-computations in figure 1. it is possible for the same word to admit two $n$-computations of different values. This comes from the fact that informal statements of the type: "few + few = few" cannot be true in absolute terms for a fixed limit value $n$. On the other hand, it is possible to show that the results of different computations do not contradict each other, modulo a certain approximation. We will in fact dismantle a slightly stronger result, involving variations on the notion of computation.

**Definition 4.** *A $n$-under-computation over word $u$ is an ordered tree of unfixed arity, each node $x$ of which is labeled by an element $v(x) \in M$ called its value and such that:*

– *the leaf labels read from left to right form a word $\leqslant u$ (the order on $M$ is extended to words of $M^*$ letter by letter),*
– *for each node $x$, with children $y_1, \ldots, y_k$ (from left to right), one of these situations is true:*
  - *$k = 0$ (i.e., $x$ is a leaf),*
  - *$k = 2$, and $v(x) \leqslant v(y_1) \cdot v(y_2)$,*
  - *$2 < k \leqslant n$, and $v(x) \leqslant v(y_1) = \cdots = v(y_k) = v(x) \in E(M)$,*
  - *$k > n$, $v(y_1) = \cdots = v(y_k) = e \in E(M)$ and $v(x) \leqslant e^\sharp$.*

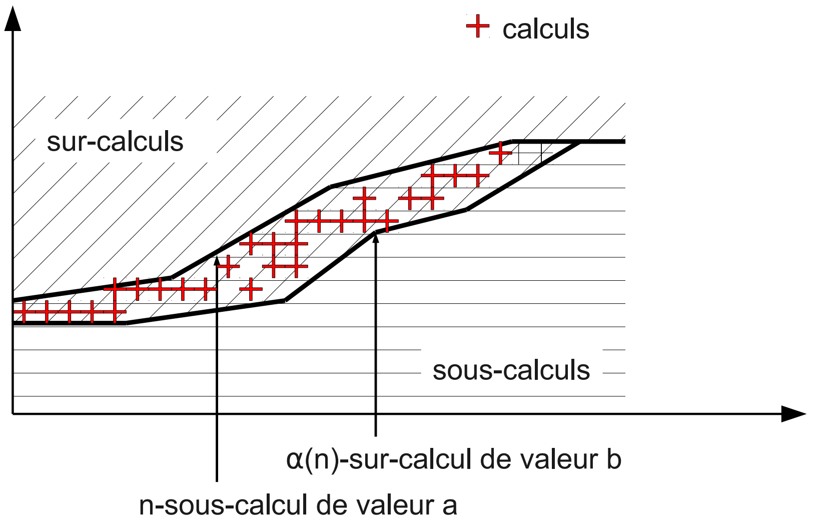
*An $n$-over-computation is defined identically by replacing $\leqslant$ by $\geqslant$ everywhere.*
*The value of an over/undercomputation tree is the value of its root.*

Idea: under-computations allow to under-approximate the value of a word, over-computations allow to over-approximate it. Under-computations and over-computations have properties that computations do not have and that make them easier to handle, as shown by the following fact:

**Fact 11** *Every $n$-computation is an $n$-under-computation and an $n$-over-computation.*

*Let $m \leqslant n$, then every $m$-under-computation is an $n$-under-computation, and every $n$-over-computation is a $m$-over-computation.*

This is interpreted in the following figure.



A word is fixed. The abscissas correspond to the value of $n$, and the ordinates to the (idealized) order in the stabilization monoid. The $n$-computations of value $a$ (which are assumed to have a constant depth equal, for example, to $3|M|$), are symbolized by red crosses at the abscissa $n$ and the ordinate $a$. The under-computations, which are closed downwards and to the right according to the previous lemma, delimit the horizontally striped zone. The over-computations, are themselves closed towards the left and the top, and correspond to the obliquely striped zone. The following lemma states that the boundaries of these zones, in bold on the figure, cannot be arbitrarily far apart; they behave, globally, in an identical manner.

**Lemma 7 (admitted).** *For all $p$, there exists $\alpha$ such that for all $\alpha(n)$-over-computation of value $b$ on a word $u$ of depth at most $p$ and all $n$-under-computation on the same word of value $a$ of depth at most $p$, $a \leqslant b$.*

Thus, the depth must be large enough for there to exist a computation by Lemma 6, but bounded so that a unique $\alpha$ allows the margin of error to be bounded by Lemma 7.

*Example 12.* Building computation trees on stabilization monoids example.

*Example 13.* A monoid $\langle M, \cdot \rangle$ can be transformed into a stabilization monoid $\langle M, \cdot, id, = \rangle$.

Then, for all $n$, an $n$-computation (resp. over-/under-computation) on a word $u$ always has the value $\pi(u)$.

**Recognizability**

Let $M$ be a stabilization monoid. An *ideal* of $M$ is a subset $I \subseteq M$ such that $a \leqslant b \in I$ implies $a \in I$.

**Lemma 8.** *Let a stabilization monoid* $\mathbf{M}$ *a map* $h : \mathbb{A} \to M$*, and an ideal* $I \subseteq M$*, we set:*

$$[\![\mathbf{M}, h, I]\!]_p^+(u) = \sup\{n+1 \ : \ \textit{there exists an } n\textit{-over-computation of value in } I \textit{ of prof at most } p \textit{ on } h(u)\}$$
$$[\![\mathbf{M}, h, I]\!]_p^-(u) = \inf\{n \ : \ \textit{there exists an } n\textit{-under-computation of value in } M \setminus I \textit{ of prof at most } p \textit{ on } h(u)\}$$

*Then:*

- *The functions* $[\![\mathbf{M}, h, I]\!]_p^+$ *and* $[\![\mathbf{M}, h, I]\!]_p^-(u)$ *are increasing (as a function of the input word, ordered letter-by-letter),*
- *If* $p \geqslant 3|M|$*, then* $[\![\mathbf{M}, h, I]\!]_p^- \leqslant [\![\mathbf{M}, h, I]\!]_p^+$*,*
- *For all* $p$*, there exists* $\alpha$ *such that* $[\![\mathbf{M}, h, I]\!]_p^+ \leqslant \alpha([\![\mathbf{M}, h, I]\!]_p^-)$*.*
- *For all* $p \leqslant r$*,* $[\![\mathbf{M}, h, I]\!]_r^- \leqslant [\![\mathbf{M}, h, I]\!]_p^-$ *and* $[\![\mathbf{M}, h, I]\!]_p^+ \leqslant [\![\mathbf{M}, h, I]\!]_r^+$*.*

*Proof.* The first item uses the Theorem 11 and the fact that $I$ is an ideal. The second item is a consequence of Lemma 6 in combination with the Theorem 11: the existence of an $n$-computation proves the inequality. The third item is a consequence of the Lemma 7: under-computations and over-computations do not contradict each other. The last item corresponds to the fact that the larger $p$ is, the more over/under-computations of depth at most $p$.

Thus, all functions $[\![\mathbf{M}, h, I]\!]_p^+$ and $[\![\mathbf{M}, h, I]\!]_p^-$ are equivalent for $\approx$. Their equivalence class is called *cost function recognized by* $\mathbf{M}, h, I$.

## 4.3 ♯-expression

A *♯-expression* on $A$ is an expression composed of letters in $A$, products, and exponent by ♯. A ♯-expression on a stabilization monoid $\mathbf{M}$ is a ♯-expression on $M$. It is *well-typed* if it denotes a valid computation in the stabilization monoid, that is, it is possible to evaluate it in $\mathbf{M}$ using the product and stabilization operations of $\mathbf{M}$. The result of this computation is called the *value* of the ♯-expression. A ♯-expression is *strict* if it contains at least one occurrence of ♯.

The *n-unfolding* is defined by induction as:

$$\begin{aligned}
\text{unfolding}(a, n) &= a \\
\text{unfolding}(EF, n) &= \text{unfolding}(E, n)\text{unfolding}(F, n) \\
\text{unfolding}(E^\sharp) &= \overbrace{\text{unfolding}(E, n) \dots \text{unfolding}(E, n)}^{n+1 \text{ times}}
\end{aligned}$$

In the case where $E$ is well-typed, for $n \geqslant 3$, the $n$-computation of $E$ is defined by induction as the computation tree

$$\begin{aligned}
\text{computation}(a, n) &= a \\
\text{computation}(EF, n) &= [v(EF)](\text{computation}(E, n), \text{computation}(F, n)) \\
\text{computation}(E^\sharp) &= [v(E^\sharp)]\overbrace{(\text{computation}(E, n), \dots, \text{computation}(E, n))}^{n+1 \text{ fils}}
\end{aligned}$$

Of course, computation$(E, n)$ is an $n$-computation on the word unfolding$(u, n)$, with value $v(E)$. In fact, it's a bit stronger:

**Fact 12** computation$(E, n)$ *is an* $m$*-computation on the word* unfolding$(E, n)$ *for all* $m \leqslant n$*.*

## 4.4 Product of stabilization monoids

Let $M = \langle M, \cdot, \sharp, \leqslant \rangle$ and $M' = \langle M', \cdot', \sharp', \leqslant' \rangle$ be two stabilization monoids. Their product is defined as:

$$M \times M' = \langle M \times M', \cdot'', \sharp'', \leqslant'' \rangle$$

where $(a, a') \cdot'' (b, b') = (a \cdot b, a' \cdot' b')$, $(e, e')^{\sharp''} = (e^{\sharp}, e'^{\sharp'})$, and $(a, a') \leqslant'' (b, b')$ iff $a \leqslant b$ and $a' \leqslant' b'$.

Given two words $u, u'$ on $M$ and $M'$ respectively, of the same length, we define $u \times u'$ as the word of $M \times M'$ which projected onto $M$ and $M'$ gives $u$ and $u'$ respectively.

**Lemma 9.** *If there exists an $n$-computation (resp. under-computation, resp. over-computation) on $u \times v$ of value $(a, b)$ in $M \times M'$, then there exists an $n$-computation (resp. under-computation, resp. over-computation) on $u$ in $M$ of value $a$ and an $n$-computation (resp. under-computation, resp. over-computation) on $v$ in $M'$ of value $b$.*

*Proof.* By projection of values.

**Corollary 1.** *Let two recognizable cost functions $f, g$ be on the same alphabet $\mathbb{A}$, then there exists a stabilization monoid $\mathbf{M}$, an application of $\mathbb{A}$ in $M$ and two ideals $I, J$ such that $\mathbf{M}, h, I$ recognizes $f$ and $\mathbf{M}, h, J$ recognizes $g$.*

**Corollary 2.** *Recognizable functions are closed under* min *and* max.

## 4.5 Decidability of the existence of a bound

**Theorem 13.** *The relation $\preccurlyeq$ is decidable on recognizable cost functions.*

*Proof.* Let two regular cost functions $f, g$. By the corollary 1 there exists a monoid $M$, a map $h$ of the alphabet $\mathbb{A}$ into $M$, and two ideals $I, J$ such that $\mathbf{M}, h, I$ recognizes $f$ and $\mathbf{M}, h, J$ recognizes $g$.

We establish that $f \leqslant g$ iff for all $a \in \langle h(\mathbb{A}) \rangle^*$, $a \in I$ implies $a \in J$.

$\Leftarrow$ Suppose for all $a \in \langle h(\mathbb{A}) \rangle^{\sharp}$ that $a \in I$ implies $a \in J$. Let $u$ be a word on $\mathbb{A}$, and $t$ be an $n$-computation tree on $u$, with value $a$. Of course, $a \in \langle h(\mathbb{A}) \rangle^{\sharp}$, and so if $a \in I$ then $a \in J$. We deduce that:

$$f \approx \sup\{n \ : \ n\text{-computation on } u \text{ with value } a \in I\} \leqslant \sup\{n \ : \ n\text{-computation on } u \text{ with value } a \in J\} \approx g.$$

$\Rightarrow$ Suppose there exists $a \in \langle h(\mathbb{A}) \rangle^{\sharp}$ such that $a \in I$ but $a \notin J$. There exists a well-typed $\sharp$-expression $E$ of value $a$ on the alphabet $h(\mathbb{A})$. Let $E'$ be obtained from $E$ by substituting for every $x \in \mathbb{A}$ a letter $b$ such that $h(b) = x$. Now, computation$(E, n)$ is an $n$-computation on $u_n = \text{unfolding}(E, n) = h(\text{unfolding}(E', n))$ of depth $P$ and value $a \in I$. So $[\![\mathbf{M}, h, I]\!]^+_P(u_n) \geqslant n$. But computation$(E, n)$ is also a $0$-computation on $u_n$, with value $a \notin J$. So $[\![\mathbf{M}, h, J]\!]^+_P(u_n) = 0$.

## 4.6 Closure under inf-projection: the monoid of ideals

Let $\mathcal{I}$ be the set of ideals on $M$. We equip $\mathcal{I}$ with a structure order by:

$$A \leqslant B \qquad \text{if and only if} \quad A \subseteq B\ ,$$

of a product by:

$$A \cdot B \quad = \quad \{c \leqslant a \cdot b \ : \ a \in A,\ b \in B\}$$

and a stabilization operator by:

$$E^{\sharp} = \{x \leqslant a \cdot e^{\sharp} \cdot b \ : \ a, b, e \in E,\ e \cdot e = e\}\ .$$

**Lemma 10.** *If $E$ is idempotent, then all $a \in E$ is such that $a \leqslant b \cdot e \cdot c$ where $b, c, e \in E$, $e$ is idempotent.*

*Proof.* Since $E = E \cdots E$, for all $n$, there exist $a_1, \ldots, a_n \in E$ such that $a \leqslant a_1 \cdots a_n$. By Ramsey, for $n$ large enough, there exist $i, j$ such that $a_i \cdots a_{j-1} = e$ is idempotent and $i > 1$. We then set $b = a_1 \cdots a_{i-1}$, $c = a_j \cdots a_n$. We have $b, c, e \in E$, and $a \leqslant b \cdot e \cdot c$.

**Lemma 11.** *$M_{\mathcal{I}} = \langle \mathcal{I}, \cdot, \sharp, \subseteq \rangle$ is a stabilization monoid.*

**Lemma 12.** *For any idempotent ideal $E$, and any under-computation $t$ on a word $u \in E^+$ of value $a$, $a \in E$. If $t$ contains a stabilization node, then $a \in E^\sharp$.*

*Proof.* By induction on the depth of the computation.

**Lemma 13.** *Any $n$-computation of depth $p$ on a word of length at least $n^p + 1$ contains a stabilization node.*

*Proof.* If the computation does not contain any stabilization node, then all its nodes have arity at most $n$. Therefore, such a computation of depth $p$ cannot have more than $n^p$ leaves. Contradiction.

**Lemma 14.** *Let $A_1 \ldots A_k$ be a word on $\mathcal{I}$ and an $n$-computation $T$ on $A_1 \ldots A_k$ of depth $p$ and value $A$. For any $a \in A$, there exists an $n$-under-computation of depth $3p$ of value $a$ on a word $a_1 \ldots a_k$ such that $a_1 \in A_1, \ldots, a_k \in A_k$.*

*Proof.* We study the case of depths 0 and 1 first.

**sheet** $T = [A_1]$ Let $a \in A = A_1$, $[a]$ is an $n$-computation of value $a \in A$.
**product** $T = [A]([A_1] \cdot [A_2])$. Let $a \in A$, by definition of the product, there exist $a_1 \in A_1$ and $a_2 \in A_2$ such that $a \leqslant a_1 \cdot a_2$. The $n$-under-computation $[a]([a_1] \cdot [a_2])$ is suitable.

**idempotent** $T = [E](\overbrace{[E] \ldots [E]}^{k})$ with $k \leqslant n$. Let $a \in E$. We have $a \leqslant b \cdot e \cdot c$ for $b, c, e \in E$, $e$ idempotent (Lemma 10). We construct the $n$-under-computation $[a]([b \cdot e]([b] \cdot [e](\overbrace{[e] \ldots [e]}^{k-2})) \cdot [c])$.

**stabilization** $T = [E^\sharp](\overbrace{[E] \ldots [E]}^{k})$ (with $k > n$). Let $a \in E^\sharp$. $a$ is written as $a \leqslant b \cdot e^\sharp \cdot c$ for $b, e, c \in E$, $e$ idempotent. We construct the $n$-under-computation$[a]([b \cdot e^\sharp]([b] \cdot [e^\sharp](\overbrace{[e] \ldots [e]}^{k-2})) \cdot [c])$.

The general case is obtained by a recursive application of the previous cases.

**Lemma 15.** *Let $A_1 \ldots A_k$ be a word on $\mathcal{I}$ and an $n^{3|M|}$-computation $T$ on $A_1 \ldots A_k$ of depth $p$ with value $A$. For any word $u = a_1 \ldots a_k \in A_1 \ldots A_k$ (i.e. $a_1 \in A_1$, $\ldots$, $a_k \in A_k$), there exists an $n$-computation on $a_1 \ldots a_k$ of value $a \in A$ and depth at most $3|M|p$.*

*Proof.* Let us first consider the case of computations of depth 0 and 1. There are 4 cases:

**sheet** $T = [A]$, and $u = a \in A$, so $[a]$ is a suitable computation.
**product** $T = [A]([B] \cdot [C])$ in which case, $u = bc$ for $b \in B$ and $c \in C$. We construct the $n$-computation $[b \cdot c]([b] \cdot [c])$ with value $b \cdot c \in A$.

**idempotent** $T = [E](\overbrace{[E] \ldots [E]}^{k})$ (for $k \leqslant n^{3|M|}$). Let $t$ be an $n$-computation on $u = a_1 \ldots a_k$ of depth at most $3|M|$ with value $a$ (it exists by Lemma 6). This tree satisfies $a \in E$ by Lemma 12.

**stabilization** $T = [E^\sharp](\overbrace{[E] \ldots [E]}^{k})$ for $k > n^{3|M|}$. Let $t$ be an $n$-computation on $u = a_1 \ldots a_k$ of depth at most $3|M|$ with value $a$ (it exists by Lemma 6). According to 13, it has a stabilization node. So its value is in $E^\sharp$ according to Lemma 12.

The proof for any $p$ is by induction on the depth of the computation $T$, applying the 4 cases above.

**Theorem 14.** *Recognizable cost functions are closed under inf-projection.*

*Proof.* Let the function $f$ on the alphabet $\mathbb{A}$ be recognized by $\mathbf{M}, h, I$. Let $z$ be an application from $\mathbb{A}$ to $\mathbb{B}$. We construct the stabilization monoid of the ideals $M_{\mathcal{I}}$, the application $H$ sending $b \in \mathbb{B}$ to $\{x \ : \ x \leqslant h(a) \ : \ z(a) = b\}$, and considering the ideal $K$ of $M_{\mathcal{I}}$:

$$K = \{J \in \mathcal{I} \ : \ J \subseteq I\} \ .$$

It is a question of showing that this construction is correct. Let $F$ be the function recognized by $M_{\mathcal{I}}, H, K$.

Let $u$ be a word on $\mathbb{B}$, and $n = F(u)$, i.e.,

$$F(u) = \inf\{n \ : \ \text{there exists an } n\text{-under-computation on } H(u) \text{ with value } A \not\subseteq I\} \ .$$

Consider the value $A \not\subseteq I$ testifying to the value $n$. Let $a \in A \setminus I$ and $T$ be the corresponding $n$-under-computation. This $n$-under-computation can be transformed into an $n$-under-computation of $M$ of value $a$ on a word $v \in H(u)$ by the lemma 14. We deduce that $n \geqslant f(v) \geqslant \inf\{f(v) \ : \ z(v) = u\}$.

Conversely, let us fix $u$, and let $n = \inf\{f(v) \ : \ z(v) = u\}+1$ (sup-def of $f$ for $3|M|^2$). By definition, there exists $v$ such that $z(v) = u$, such that every $n$-under-computation on $h(v)$ has value $a \notin I$ (of depth $3|M|^2$). Consider an $n^P$-computation on $H(u)$ of value $A$ of height $P$. By Lemma 15, there exists an $n$-computation of value $a \in A$, and of depth at most $P^2$. We deduce that $a \notin I$. Therefore $A \in K$. Therefore $F(u) \leqslant n$.

## 4.7 From stabilization monoids to automata

We construct a B-expression that gives a word a value smaller than $n$ if there exists an $n$-under-computation on this word. The construction is by induction on the depth $p$ of the computation.

$$E_{a,0} = \emptyset \qquad\qquad E_{a,p+1} = \sum_{b \in \mathbb{A},\ h(b) \geqslant a} b + \sum_{b \cdot c \geqslant a} E_{b,p} E_{c,p} + \sum_{e = e \cdot e \geqslant a} E^{B}_{a,p+1} + \sum_{e^\sharp \geqslant a} E^{*}_{b,p+1} \ .$$

Finally, we set:

$$E = \sum_{a \notin I} E_{a,3|M|} \ .$$

We show by induction on $p$ that $E_{a,p} = [\![\mathbf{M}, h, \{x \leqslant a\}]\!]^{-}_{p}$, which amounts to establishing that:

$$u \in [\![E_{a,p}]\!](u)$$
$$\text{if and only if}$$
$$\text{there exists an } n\text{-under-computation on } \tilde{h}(u) \text{ of height at most } p \text{ of value} \geqslant a$$

from which we deduce $[\![E]\!]_B = [\![\mathbf{M}, h, I]\!]^{-}_{3|M|}$.

## 4.8 From automata to stabilization monoids

Idea: use closure under inf-projection, under max, the fact that counting the number of a's and that the characteristic function of any regular language is a regular cost function.

# 5 S-automata and duality theorem

S-automata resemble B-automata in many ways, although they are in a sense opposite to them. The results seen previously for B-automata all have a counterpart concerning S-automata. We do not prove these results in this course material. The proofs follow similar patterns, but all must be completely reconstructed.

The most important result is Theorem 17 which states that the functions computed by B-automata and by S-automata are equivalent.

## 5.1 Definition of S-automata

## 5.2 Cost of a sequence

Consider a sequence $u \in \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{c}\}^*$, we saw during the definition of B-automata the set $C(u)$ which collects the values observed during the sequence. This time, we set:

$$cost_S(u) = \min C(u) \ .$$

The cost of a sequence is the smallest value taken by the counter at the time of being tested, just before being reset to 0.

Let $\Gamma$ be a finite set of *counters*, the set of *actions on $\Gamma$* is $\mathrm{A}_\Gamma = \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^\Gamma$. Either:

$$\begin{aligned} cost_S : \quad & \mathrm{A}_\Gamma{}^* \to \omega \\ & u \to \min C(u) \end{aligned}$$

(in particular $cost_S = \omega$ if $\Gamma = \emptyset$).

An *S-automaton* $\mathcal{A} = \langle Q, A, I, F, \Gamma, \Delta\rangle$ consists of:

- a finite set of *states* $Q$,
- a finite alphabet $A$,
- a set of *initial states* $I \subseteq Q$,
- a set of *final states* $F \subseteq Q$,
- a set of *transitions* $\Delta \subseteq Q \times A \times \mathrm{A}_\Gamma \times Q$.

An run $\rho$ of $\mathcal{A}$ on $u \in A^*$ is a sequence $\rho = (p_0, a_1, t_1, p_1) \dots (p_{n-1}, a_n, t_n, p_n)$ such that:

- $(p_{i-1}, a_i, t_i, p_i) \in \Delta$,
- $a_1 \dots a_n = u$,
- $p_0 \in I$,
- $p_n \in F$.

Its cost is $cost_S(\rho) = cost_S(t_1 \dots t_n)$.

We define the *cost* of a word $u \in A^*$ for $\mathcal{A}$ as:

$$[\![\mathcal{A}]\!]_S(u) = \max\{cost_S(\rho) \ : \ \rho \text{ run of } \mathcal{A} \text{ on } u\} \qquad \text{(computed in } \omega+1).$$

*Remark 7.* If there is no run of $\mathcal{A}$ on a word $u$, then $[\![\mathcal{A}]\!]_S(u) = 0$.

*Remark 8.* If $\mathcal{A}$ is a S-automaton without a counter (i.e., $\Gamma = \emptyset$) then $\mathcal{A}$ can be seen as a non-deterministic automaton accepting the language $L$. We then have for any word $u$:

$$[\![\mathcal{A}]\!]_S(u) = \begin{cases} \omega & \text{if } u \in L, \\ 0 & \text{otherwise.} \end{cases}$$

In particular $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ if and only if $[\![\mathcal{A}]\!]_S \leqslant [\![\mathcal{A}]\!]_S$.

This situation denotes a behavior of S-automata opposite to that of B-automata.

*Example 14.* Number of $a$.

*Example 15.* Max of the number of $a$ and $b$.

*Example 16.* Min of the number of $a$ and $b$.

*Example 17.* Size of the smallest maximal segment formed only by $a$.

### 5.3 Elementary results

The elementary results are similar to the case of B-automata, the constructions are identical.

**Proposition 11.** *The support of a function defined by B-automata is regular (i.e., the set of finite-valued words).*

**Proposition 12.** *The functions computed by B-automata are closed under* min, max *and sup-projection (let $f$ be a letter-to-letter morphism, and $\mathcal{A}$ a B-automata, there exists $\mathcal{A}'$ such that $[\![\mathcal{A}']\!]_{SS}(u) = \sup\{[\![\mathcal{A}]\!]_S(v) \;:\; f(v) = u\}$).*

### 5.4 Equivalences

As for the B-automata, there are several formalisms equivalent to the S-automata modulo $\approx$.

**Theorem 15.** *The following properties are indeed equivalent:*

1. *$f$ is computed by a S-automata (with or without $\epsilon$-transitions) modulo $\approx$,*
2. *$f$ is computed by a $S$-hierarchical automata (with or without $\epsilon$-transitions) modulo $\approx$,*
3. *$f$ is computed by an $S$-regular expression modulo $\approx$ or is bounded.*

The definitions of hierarchical S-automata, and of S-automata with $\epsilon$-transitions are similar to those of B-automata (we must replace inf with sup in the latter case). It remains to define $S$-regular expressions.

An *S-regular expression* (or *S-expression*) respects the following syntax:

$$E ::= \quad \varepsilon \quad | \quad a \quad | \quad E+E \quad | \quad E \cdot E \quad | \quad E^* \quad | \quad E^S \quad (|\emptyset) \;.$$

The semantics $[\![e]\!]_{SS}$ of an S-regular expression $e$ is an application of $\mathbb{A}^*$ in $\omega+1$ defined by induction:

- $[\![\varepsilon]\!]_{SS}(u) = \begin{cases} \omega & \text{if } u = \varepsilon \\ 0 & \text{otherwise} \end{cases}$,
- $[\![a]\!]_S(u) = \begin{cases} \omega & \text{if } u = a \\ 0 & \text{otherwise} \end{cases}$,
- $[\![f+g]\!]_S(u) = \max([\![f]\!]_S(u), [\![g]\!]_S(u))$,
- $[\![f \cdot g]\!]_{SS}(u) = \max\limits_{u=vw} \min([\![f]\!]_S(u), [\![g]\!]_S(v))$,
- $[\![f^*]\!]_S(u) = \max\limits_{u=u_1\ldots u_n} \min\limits_{i=1\ldots n} [\![f]\!]_S(u_i)$,
- $[\![f^S]\!]_S(u) = \max\limits_{u=u_1\ldots u_n} \min(n, \min\limits_{i=1\ldots n} [\![f]\!]_S(u_i))$.

**Proposition 13.** *If $e$ is a regular expression describing the language $L$, then*

$$[\![e]\!]_S(u) = \begin{cases} \omega & \textit{if } u \in L \\ 0 & \textit{otherwise.} \end{cases}$$

*Example 18.* $[\![a^S]\!]_S = [\![a^B + (a^*b)^+a^*]\!]_B$, $[\![(a^Sb)^*]\!]_S = [\![(a^*b)^*a^Bb(a^*b)]\!]_B$,

**Theorem 16.** *The existence of a bound is decidable for functions computed by S-automata.*

*Proof.* We start with an $S$-regular expression. We simplify the $\emptyset$. If the result is $\emptyset$, then the expression is bounded.

### 5.5 Duality theorem, and regular cost functions

The core of the theory is based on the following theorem.

**Theorem 17 (duality [5]).** *The following properties are equivalent for a function $f$ from $\mathbb{A}^*$ to $\omega+1$:*

- *$f$ is computed by a B-automaton modulo $\approx$,*
- *$f$ is computed by a S-automaton modulo $\approx$.*

The proof of this theorem requires going through the equivalent model of cost functions recognizable by stabilization monoid. It is not given in these notes.

## 6 Conclusion/summary

In this course, we have seen what cost functions are, as well as different acceptor models for defining them:

- B-automata, with or without $\varepsilon$-transitions, hierarchical or not,
- B-regular expressions,
- S-automata, with or without $\varepsilon$-transitions,
- S-regular expressions,
- stabilization monoids,
- (not present in these notes) monadic cost logic.

All these models are indeed equivalent, and these equivalences are natural extensions of classical results on regular finite-word languages. The class obtained is that of "regular cost functions".

Finally, this class also has good decidability properties; in particular, the domination between regular cost functions is decidable, and so in particular the existence of a bound is also decidable.

A more complete presentation of this theory would require presenting more general forms of it:

- on infinite words of length $\omega$, or more generally of countable length,
- on finite trees,
- in a limited way on infinite trees (the general problem remaining open),
- the extension to several orders of magnitude on finite words,
- the links with non-standard analysis,

as well as its applications,

- to the (restricted) star height,
- to the separation of regular languages by formulas of $\Sigma_2$,
- to the star height on trees,
- to the decidability of the Mostowski index on finite trees (open in the general case),
- to the finite power property, and more generally the existence of a bound on the iteration of a monadic fixed point,
- to the resolution of games with promises,
- to the resolution of quantitative variants of Church's synthesis problem,
- to the links with MSO+U logic.

Most of these questions are addressed in [6].

## References

1. Sebastian Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 596–607. Springer, 2004.
2. Achim Blumensath, Martin Otto, and Mark Weyer. Boundedness of monadic second-order formulae over finite words. In *36th ICALP*, Lecture Notes in Computer Science, pages 67–78. Springer, July 2009.
3. Mikołaj Bojańczyk. Factorization forests. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2009.
4. Thomas Colcombet. Factorisation forests for infinite words. In *FCT 07*, number 4639 in Lecture Notes in Computer Science, pages 226–237, Budapest, August 2007. Springer.
5. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *36th ICALP*, number 5556 in Lecture Notes in Computer Science, pages 139–150, Rhodos, July 2009. Springer.
6. Thomas Colcombet. Fonctions de coût régulières. Habilitation à diriger des recherches, 2013.
7. Françoise Dejean and Marcel-Paul Schützenberger. On a question of eggan. *Information and Control*, 9(1):23–25, 1966.
8. L. C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10:385–397, 1963.
9. Kosaburo Hashiguchi. A decision procedure for the order of regular events. *Theoretical Computer Science*, 8:69–72, 1979.
10. Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.
11. Kosaburo Hashiguchi. Regular languages of star height one. *Information and Control*, 53(3):199–210, 1982.
12. Kosaburo Hashiguchi. Representation theorems on regular languages. *J. Comput. Syst. Sci.*, 27(1):101–115, 1983.
13. Kosaburo Hashiguchi. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*, pages 74–88, 1988.
14. Daniel Kirsten. Desert automata and the finite substitution problem. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.
15. Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.
16. Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Internat. J. Algebra Comput.*, 4(3):405–425, 1994.
17. Manfred Kufleitner. The height of factorization forests. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *MFCS 2008: Mathematical Foundations of Computer Science*, volume 5162 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 2008.
18. Robert McNaughton. The loop complexity of pure-group events. *Information and Control*, 11(1-2):167–176, 1967.
19. Imre Simon. Limited subsets of a free monoid. In *FOCS*, pages 143–150. IEEE, 1978.
20. Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72:65–94, 1990.