Logical Methods in Computer Science
Vol. 10(1:4)2014, pp. 1–37
www.lmcs-online.org

Submitted Feb. 22, 2013
Published Feb. 4, 2014

LINEAR TEMPORAL LOGIC FOR REGULAR COST FUNCTIONS

DENIS KUPERBERG

Liafa/CNRS/Université Paris 7, Denis Diderot, France
e-mail address: denis.kuperberg@gmail.com

Abstract. Regular cost functions have been introduced recently as an extension to the notion of regular languages with counting capabilities, which retains strong closure, equivalence, and decidability properties. The specificity of cost functions is that exact values are not considered, but only estimated.

In this paper, we define an extension of Linear Temporal Logic (LTL) over finite words to describe cost functions. We give an explicit translation from this new logic to two dual form of cost automata, and we show that the natural decision problems for this logic are PSPACE-complete, as it is the case in the classical setting. We then algebraically characterize the expressive power of this logic, using a new syntactic congruence for cost functions introduced in this paper.

1. Introduction

Since the seminal works of Kleene and Rabin and Scott, the theory of regular languages is one of the cornerstones in computer science. Regular languages have many good properties, of closure, of equivalent characterizations, and of decidability, which makes them central in many situations.

Recently, the notion of regular cost function for words has been presented as a candidate for being a quantitative extension to the notion of regular languages, while retaining most of the fundamental properties of the original theory such as the closure properties, the various equivalent characterizations, and the decidability [Col09]. A cost function is an equivalence class of the functions from the domain (words in our case) to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation $\approx$ which allows some distortion, but preserves the boundedness property over each subset of the domain. The model is an extension to the notion of languages in the following sense: one can identify a language with the function mapping each word inside the language to 0, and each word outside the language to ∞. It is a strict extension since regular cost functions have counting capabilities, e.g., counting the number of occurrences of letters, measuring the length of intervals, etc...

This theory grew out of two main lines of work: research by Hashiguchi [Has82], Kirsten [Kir05], and others who were studying problems which could be reduced to whether or not

2012 ACM CCS: [**Theory of computation**]: Models of computation; Formal languages and automata theory.

Key words and phrases: LTL, cost functions, cost automata, stabilization semigroup, aperiodic, syntactic congruence.

DOI:10.2168/LMCS-10(1:4)2014

some function was bounded over its domain (the most famous of these problems being the star height problem); and research by Bojańczyk and Colcombet [Boj04, BC06] on extensions of monadic second-order logic (MSO) with a quantifier U which can assert properties related to boundedness.

Linear Temporal Logic (LTL), which is a natural way to describe logical constraints over a linear structure, have also been a fertile subject of study, particularly in the context of regular languages and automata [VW86]. Moreover quantitative extensions of LTL have recently been successfully introduced. For instance the model Prompt-LTL introduced in [KPV09] is interested in bounding the waiting time of all requests of a formula, and in this sense is quite close to the aim of cost functions.

In this paper, we extend LTL (over finite words) into a new logic with quantitative features ($\mathrm{LTL}^{\leq}$), in order to describe cost functions over finite words with logical formulae. We do this by adding a new operator $U^{\leq N}$: a formula $\phi U^{\leq N} \psi$ means that ψ holds somewhere in the future, and ϕ has to hold until that point, except at most N times (we allow at most N "mistakes" of the Until formula). The variable N is unique in the formula, and the semantic of the formula is the least value of N which makes the statement true.

Related works and motivating examples. Regular cost functions are the continuation of a sequence of works that intend to solve difficult questions in language theory. Among several other decision problems, the most prominent example is the star-height problem: given a regular language L and an integer k, decide whether L can be expressed using a regular expression using at most k-nesting of Kleene stars. The problem was resolved by Hashigushi [Has88] using a very intricate proof, and later by Kirsten [Kir05] using an automaton that has counting features.

Finally, also using ideas inspired from [BC06], the theory of those automata over words has been unified in [Col09], in which cost functions are introduced, and suitable models of automata, algebra, and logic for defining them are presented and shown equivalent. Corresponding decidability results are provided. The resulting theory is a neat extension of the standard theory of regular languages to a quantitative setting.

On the logic side, Prompt-LTL, introduced in [KPV09], and PLTL [AETP01], which are similar, show an interesting way to extend LTL in order to look at boundedness issues, and already gave interesting decidability and complexity results. In [DJP04], the logics kTL was introduced, which uses an explicit bound k to express some desired boundedness properties.

These logics are only interested in bounding the wait time, i.e. consecutive events. It would correspond in the framework of regular cost functions to the subclass of temporal cost functions introduced in [CKL10].

We will introduce here a logic $\mathrm{LTL}^{\leq}$ with a more general purpose : it can bound the wait time before an event, but also non-consecutive events, like the number of occurences of a letter in a word.

These quantitative issues are a quite natural preoccupation in the context of verification: for instance one would expect that a system can react in a bounded time. The new features of $\mathrm{LTL}^{\leq}$ could possibly be used to allow some mistakes in the behaviour fo the program, but guarantee a global bound on the number of mistakes. An other issue is the consumption of resources: for instance it is interesting to know whether we can bound the number of times a program stores something in the memory.

Contributions. It is known from [Col09] that regular cost functions are the ones recognizable by stabilization semigroups (or in an equivalent way, stabilization monoids), and from [CKL10] than there is an effective quotient-wise minimal stabilization semigroup for each regular cost function. This model of semigroups extends the standard approach for languages.

We introduce a quantitative version of LTL in order to describe cost functions by means of logical formulas. The idea of this new logic is to bound the number of "mistakes" of Until operators, by adding a new operator $U^{\leq N}$. The first contribution of this paper is to give a direct translation from LTL$^{\leq}$-formulae to B-automata, which is an extension of the classic translation from LTL to Büchi automaton for languages. This translation preserves exact values (i.e. not only cost functions equivalence), which could be interesting in terms of future applications. We also use dual forms of logic and cost automata to describe a similar translation, and show that the boundedness problem for LTL$^{\leq}$-formulae is PSPACE-complete (as it was the case in the classical setting). Therefore, we do not lose anything in terms of computational complexity, when generalizing from LTL to LTL$^{\leq}$.

We then show that regular cost functions described by LTL formulae are the same as the ones computed by aperiodic stabilization semigroups, and this characterization is effective. The proof uses a syntactic congruence for cost functions, introduced in this paper, which generalizes the Myhill-Nerode equivalence for regular languages. This congruence present a general interest besides this particular context, since it can be used for any regular cost function.

This work validates the algebraic approach for studying cost functions, since it shows that the generalization from regular languages extends also to syntactic congruence. It also allows a more user-friendly way to describe cost functions, since temporal logic is often more intuitive than automata or stabilization semigroups to describe a given cost function.

As it was the case in [CKL10] for temporal cost functions, the characterization result obtained here for LTL$^{\leq}$-definable cost functions follows the spirit of Schützenberger's theorem, which links star-free languages with aperiodic monoids [Sch65].

Organisation of the paper. After some notations, and reminder on cost functions and stabilization semigroups, we introduce in Section 4 LTL$^{\leq}$ as a quantitative extension of LTL, and give an explicit translation from LTL$^{\leq}$-formulae to B and S-automata in Sections 5 and 6. We then present in Section 7 a syntactic congruence for cost functions, and show that it indeed computes the minimal stabilization semigroup of any regular cost function. We finally use this new tool to show that LTL$^{\leq}$ has the same expressive power as aperiodic stabilization semigroups.

Notations. We will note $\mathbb{N}$ the set of non-negative integers and $\mathbb{N}_\infty$ the set $\mathbb{N} \cup \{\infty\}$, ordered by $0 < 1 < \cdots < \infty$. We will say that a set $X \subseteq \mathbb{N}_\infty$ is bounded if there is a number $N \in \mathbb{N}$ such that for all $x \in X$, we have $x < N$. In particular, if X contains ∞ then X is unbounded. If E is a set, $E^{\mathbb{N}}$ is the set of infinite sequences of elements of E (we will not use here the notion of infinite word). Such sequences will be denoted by bold letters ($\mathbf{a}$, $\mathbf{b}$,...). We will work with a fixed finite alphabet $\mathbb{A}$. The set of words over $\mathbb{A}$ is $\mathbb{A}^*$ and the empty word will be noted ϵ. The concatenation of words u and v is uv. The length of u is $|u|$. The number of occurrences of letter a in u is $|u|_a$. We will use $|\cdot|$ (resp. $|\cdot|_a$) to note the function $u \mapsto |u|$ (resp. $u \mapsto |u|_a$). Functions $\mathbb{N} \to \mathbb{N}$ will be denoted by letters $\alpha, \beta, \ldots,$

and will be extended to $\mathbb{N} \cup \{\infty\}$ by $\alpha(\infty) = \infty$. Such functions will be called *corrections functions*.

2. Regular Cost functions

2.1. **Cost functions and equivalence.** Let $\mathcal{F}$ be the set of functions from $\mathbb{A}^*$ to $\mathbb{N}_\infty$. If $L \subseteq \mathbb{A}^*$, we will note χ_L the function of $\mathcal{F}$ defined by $\chi_L(u) = 0$ if $u \in L$, ∞ if $u \notin L$. For $f, g \in \mathcal{F}$, we say that $f \preccurlyeq g$ if for all set $W \subseteq \mathbb{A}^*$, if $g(W)$ is bounded then $f(W)$ is bounded. We define the equivalence relation $\approx$ on $\mathcal{F}$ by $f \approx g$ if $f \preccurlyeq g$ and $g \preccurlyeq f$. Notice that $f \approx g$ means that f and g are bounded on the same sets of words, i.e. for all $W \subseteq \mathbb{A}^*$, we have $f(W)$ is bounded if and only if $g(W)$ is bounded. This equivalence relation does not pay attention to exact values, but preserves the existence of bounds.

We also introduce another relation, which is parametrized by a correction function. If α is a correction function (see Notations), we say that $f \leq_\alpha g$ if $f \leq \alpha \circ g$, and $f \approx_\alpha g$ if $f \leq_\alpha g$ and $g \leq_\alpha f$. Intuitively, $f \approx_\alpha g$ means that one can be obtained from the other by "distorting" the value according to the correction function α. In particular, $f \approx_{id} g$ if and only if $f = g$ (where id is the identity function).

Lemma 2.1. [Col09] *Let $f, g \in \mathcal{F}$. We have $f \preccurlyeq g$ (resp. $f \approx g$) if and only if there exists a correction function α such that $f \leq_\alpha g$ (resp. $f \approx_\alpha g$).*

Proof. Assume $f \leq_\alpha g$ for some α. If $g(W)$ is bounded by M for some set $W \subseteq \mathbb{A}^*$, then $f(W)$ is bounded by $\alpha(M)$, so we get $f \preccurlyeq g$.

Conversely, if $f \preccurlyeq g$, we want to to build α such that $f \leq_\alpha g$. For each $n \in \mathbb{N}$, we define $W_n = \{u \in \mathbb{A}^* \mid g(u) \leq n\}$. We define $\alpha(n) = \sup f(W_n)$ if $W_n \neq \emptyset$, and $\alpha(n) = n$ otherwise. As always, $\alpha(\infty) = \infty$. Notice that because $f \preccurlyeq g$, for every $n \in \mathbb{N}$ we have $\alpha(n) \in \mathbb{N}$, since f is bounded on W_n. Let $u \in \mathbb{A}^*$. If $g(u)$ is finite, then let $n = g(u)$, we have $u \in W_n$, so $f(u) \leq \alpha(n) = \alpha \circ g(u)$. If $g(u) = \infty$, then we always have $f(u) \leq \alpha \circ g(u) = \infty$.

We showed that $f \preccurlyeq g$ if and only if there exists a correction function α such that $f \leq_\alpha g$. It directly follows that if $f \approx_\alpha g$, then $f \preccurlyeq g$ and $g \preccurlyeq f$, thus $f \approx g$. Conversely, if $f \approx g$, then there are correction function α, β such that $f \leq_\alpha g$ and $g \leq_\beta f$. We get $f \approx_{\max(\alpha,\beta)} g$. □

Notice that saying $f \approx_\alpha g$ is more precise than saying $f \approx g$: in addition to preserving the qualitative information on bound, the correction function α gives a quantitative information on the distortion of bounds.

A *cost function* is an equivalence class of $\mathcal{F}/\approx$. In practice, cost functions will always be represented by one of their elements in $\mathcal{F}$. If f is a function in $\mathcal{F}$, we will note $f^\approx$ the cost function containing f. We will say that an object (automaton, logical formula) *recognizes a cost function*, when it defines a function in $\mathcal{F}$, but the notion of equivalence we are mostly interested in is the $\approx$-equivalence instead of the equality of functions.

Notice that the value ∞ is considered unbounded, so if L and L' are languages of $\mathbb{A}^*$, then $\chi_L \approx \chi_{L'}$ if and only if $L = L'$. This shows that considering languages as cost functions does not lose any information on these languages, and therefore cost function theory properly extends language theory.

Remark 2.2. They are uncountably many cost functions in $\mathcal{F}/\approx$, and each cost function contains uncountably many functions. Therefore it is hard to give an explicit description

of all the functions in a $\approx$-class, other than all the functions equivalent to a particular representative.

We will now introduce two models of cost automata recognizing cost functions. These definitions are from [Col09], the reader can report to it for more details. In both cases, we define the semantic of an automaton $\mathcal{A}$ as a function $[\![\mathcal{A}]\!]$ in $\mathcal{F}$, which we will mainly look as a representative of the cost function $[\![\mathcal{A}]\!]^{\approx}$.

2.2. **B-automata.** A B-automaton is a tuple $\langle Q, \mathbb{A}, \mathit{In}, \mathit{Fin}, \Gamma, \Delta\rangle$ where Q is the set of states, $\mathbb{A}$ the alphabet, In and Fin the sets of initial and final states, Γ the set of counters, and $\Delta \subseteq Q \times \mathbb{A} \times \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^{\Gamma} \times Q$ is the set of transitions.

Counters have integers values starting at 0, and an atomic action $\sigma \in \{\varepsilon, \mathtt{ic}, \mathtt{r}\}^{\Gamma}$ update the value of every counter γ in the following way: `ic` increments by 1, `r` resets to 0, and ε leaves the counter value unchanged. If e is a run, let $C(e)$ be the set of values reached during e, at any point of the run and on any counter of Γ. The notation "`ic`" stands for "increment check", meaning that as soon as we increment a counter, we put its value in $C(e)$.

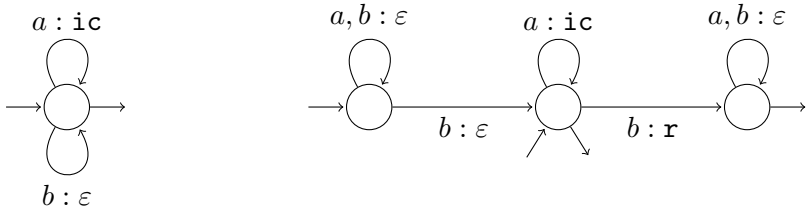
A B-automaton $\mathcal{A}$ recognizes a cost function $[\![\mathcal{A}]\!]^{\approx}_{B}$ via the following semantic:

$$[\![\mathcal{A}]\!]_B(u) = \inf\{\sup C(e), e \text{ run of } \mathcal{A} \text{ over } u\}.$$

With the usual conventions that $\sup\emptyset = 0$ and $\inf\emptyset = \infty$. It means that the value of a run is the maximal value reached by a counter, and the nondeterminism resolves in taking the run with the least value. If there is no accepting run on a word u, then $[\![\mathcal{A}]\!]_B(u) = \infty$.

Notice that in particular, if the automaton does not have any counter, then it is a classical automaton recognizing a language L, and its semantic is $[\![\mathcal{A}]\!]_B = \chi_L$, with $\chi_L(u) = 0$ if $u \in L$ and $\chi_L(u) = \infty$ if $u \notin L$.

Example 2.3. Let $\mathbb{A} = \{a, b\}$. The functions $|\cdot|_a$ and $2|\cdot|_a + 5$ represent the same cost function, which is recognized by the following one-counter B-automaton on the left-hand side. The cost function containing $u \mapsto \min\{n \in \mathbb{N},\ a^n \text{ factor of } u\}$ is recognized by the nondeterministic one-counter B-automaton on the right-hand side.



2.3. **S-automata.** The model of S-automaton is dual to the one B-automaton. The aim of this model is to mimic completation: as we cannot complement a function, we get around it by reversing the semantic of the automata defining it.

An S-automaton is a tuple $\langle Q, \mathbb{A}, \mathit{In}, \mathit{Fin}, \Gamma, \Delta\rangle$ where Q is the set of states, $\mathbb{A}$ the alphabet, In and Fin the sets of initial and final states, Γ the set of counters, and $\Delta \subseteq Q \times \mathbb{A} \times \{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^{\Gamma} \times Q$ is the set of transitions.

Counters have integers values starting at 0, and an action $\sigma \in (\{\varepsilon, \mathtt{i}, \mathtt{r}, \mathtt{cr}\}^*)^{\Gamma}$ performs a sequence of atomic actions on each counter, where atomic actions are either `i` (increment by 1), `r` (reset to 0), ε (do nothing on the counter), or `cr` (check the counter value and reset

it). If e is a run, let $C(e)$ be the set of values checked during e on all counters of Γ. This means that this time, contrary to what happened in B-automata, we only put in $C(e)$ values witnessed during an operation `cr`. This is because we will be interested in the minimum of these values, and therefore we do not want to observe all intermediate values.

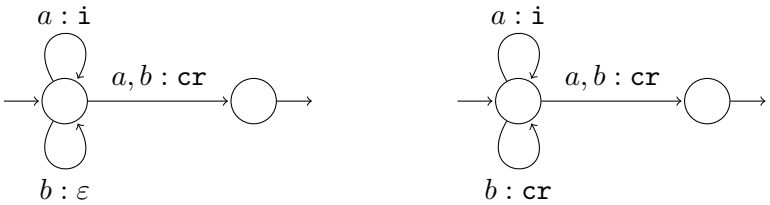
An S-automaton $\mathcal{A}$ computes a cost function $[\![\mathcal{A}]\!]_S^{\approx}$ via the following semantic :

$$[\![\mathcal{A}]\!]_S(u) = \sup\{\inf C(e), e \text{ run of } \mathcal{A} \text{ over } u\}.$$

Notice that inf and sup have been switched, compared to the definition of the B-semantic. It means that the value of a run of an S-automaton is the minimal checked value, and the automaton tries to maximize its value among all runs.

In particular, if $\mathcal{A}$ is a classical automaton for L, then its S-semantic is $[\![\mathcal{A}]\!]_S = \chi_{\overline{L}}$, where $\overline{L}$ is the complement of L. This conforts the intuition that switching between B and S-automata corresponds to complementation.

Example 2.4. We will redefine the two cost functions from example 2.3, this time with S-automata. The first one counts the number of a, and guesses the last letter to check the value. Notice that the exact function it computes is between $|\cdot|_a - 1$ and $|\cdot|_a$, so is equivalent to $|\cdot|_a$ up to $\approx_\alpha$, with $\alpha(x) = x + 1$. The second automaton counts all blocks of a, and also needs to guess the last letter, in order to count the last block (-1 if the last letter is a).



Theorem 2.5. [Col09] *If $f^{\approx}$ is a cost function, there is a B-automaton for $f^{\approx}$ if and only if there is an S-automaton for $f^{\approx}$. That is to say, B and S-automata have same expressive power (up to $\approx$) in term of recognized cost functions.*

3. Stabilization semigroups

3.1. **Classical ordered semigroups, and regular languages.** An ordered semigroup is a tuple $\mathbf{S} = \langle S, \cdot, \leq \rangle$, where $\cdot$ is a product $S \times S \to S$, and $\leq$ is a partial order compatible with $\cdot$, i.e. $\forall x, y, z \in S, x \leq y$ implies $z \cdot x \leq z \cdot y$ and $x \cdot z \leq y \cdot z$. We will always write $\mathbf{S}$ for the whole structure, and S for the underlying set.

An *ideal* of $\mathbf{S}$ is a set $I \subseteq S$ which is $\leq$-closed, i.e. such that for all $x \in I$ and $y \leq x$, we have $y \in I$.

We recall how a classical semigroup can recognize a regular language $L \subseteq \mathbb{A}^*$. The order is not necessary here. Let $h : \mathbb{A} \to S$ be a function, canonically extended to a morphism $h : \mathbb{A}^+ \to S$. Let $P \subseteq S$ be a subset of S, called *accepting subset*.

Then the language recognized by $\mathbf{S}, h, P$ is $L = h^{-1}(P)$. It is well-known that a language is regular if and only if it can be recognized by a finite semigroup.

This section explains how to generalize this to the cost functions setting, as it was done in [Col09].

3.2. **Cost sequences.** The aim is to give a semantic to stabilization semigroups. Some mathematical preliminaries are required.

Let $(E, \leq)$ be an ordered set, α a function from $\mathbb{N}$ to $\mathbb{N}$, and $\mathbf{a}, \mathbf{b} \in E^{\mathbb{N}}$ two infinite sequences. We define the relation $\preceq_\alpha$ by $\mathbf{a} \preceq_\alpha \mathbf{b}$ if :

$$\forall n. \forall m. \quad \alpha(n) \leq m \to \mathbf{a}(n) \leq \mathbf{b}(m) \ .$$

A sequence $\mathbf{a}$ is said to be *α-non-decreasing* if $\mathbf{a} \preceq_\alpha \mathbf{a}$. We define $\sim_\alpha$ as $\preceq_\alpha \cap \succeq_\alpha$, and $\mathbf{a} \preceq \mathbf{b}$ (resp. $\mathbf{a} \sim \mathbf{b}$) if $\mathbf{a} \preceq_\alpha \mathbf{b}$ (resp. $\mathbf{a} \sim_\alpha \mathbf{b}$) for some α.

Remarks:

- if $\alpha \leq \alpha'$ then $\mathbf{a} \preceq_\alpha \mathbf{b}$ implies $\mathbf{a} \preceq_{\alpha'} \mathbf{b}$,
- if $\mathbf{a}$ is α-non-decreasing, then it is α-equivalent to a non-decreasing sequence,
- $\mathbf{a}$ is id-non-decreasing iff it is non-decreasing,
- let $\mathbf{a}, \mathbf{b} \in E^{\mathbb{N}}$ be two non-decreasing sequences, then $\mathbf{a} \preceq_\alpha \mathbf{b}$ iff $\mathbf{a} \circ \alpha \leq \mathbf{b}$.

The α-non-decreasing sequences ordered by $\preceq_\alpha$ can be seen as a weakening of the $\alpha = id$ case. We will identify the elements $a \in E$ with the constant sequence of value a.

The relations $\preceq_\alpha$ and $\sim_\alpha$ are not transitive, but the following property guarantees a certain kind of transitivity.

Fact 3.1. $\mathbf{a} \preceq_\alpha \mathbf{b} \preceq_\alpha \mathbf{c}$ implies $\mathbf{a} \preceq_{\alpha \circ \alpha} \mathbf{c}$ and $\mathbf{a} \sim_\alpha \mathbf{b} \sim_\alpha \mathbf{c}$ implies $\mathbf{a} \sim_{\alpha \circ \alpha} \mathbf{c}$.

The function α is used as a "precision" parameter for $\sim$ and $\preceq$. Fact 3.1 shows that a transitivity step costs some precision. For any α, the relation $\preceq_\alpha$ coincides over constant sequences with order $\leq$ (up to identification of constant sequences with their constant value). Consequently, the infinite sequences in $E^{\mathbb{N}}$ ordered by $\preceq_\alpha$ form an extension of $(E, \leq)$.

In the following, while using relations $\preceq_\alpha$ and $\sim_\alpha$, we may forget the subscript α and verify instead that the proof has a bounded number of transitivity steps.

Definition 3.2. Let $\langle S, \cdot, \leq \rangle$ be an ordered semigroup and I be an ideal of S.

- If $\mathbf{a} \in S^{\mathbb{N}}$ is an α-non-decreasing sequence of elements of S, we note
$$I[\mathbf{a}] = \inf \{ n \in \mathbb{N} : \mathbf{a}(n) \notin I \} .$$
In other words, $I[\mathbf{a}]$ is the first position where $\mathbf{a}$ gets out of I.
- If $x, y \in S$ and $m \in \mathbb{N}$, we define the cost sequence $x|_m y$ by $(x|_m y)(n) = \begin{cases} x \text{ if } n \leq m \\ y \text{ otherwise} \end{cases}$.

3.3. **Stabilization semigroups.** The notion of stabilization semigroup is introduced in [Col09], in order to extend the classic notion of semigroups, and recognize cost functions instead of languages. If $\mathbf{S} = \langle S, \cdot \rangle$ is a semigroup (possibly with other operations), we will note $E(\mathbf{S})$ the set of idempotent elements of $\mathbf{S}$, i.e. elements $e \in S$ such that $e \cdot e = e$.

Definition 3.3. A *stabilization semigroup* $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ is an ordered semigroup $\langle S, \cdot, \leq \rangle$ together with an operator $\sharp$: $E(\mathbf{S}) \to E(\mathbf{S})$ (called *stabilization*) such that:

- for all $a, b \in S$with $a \cdot b \in E(\mathbf{S})$ and $b \cdot a \in E(\mathbf{S})$, $(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b$;
- for all $e \in E(\mathbf{S})$, $(e^\sharp)^\sharp = e^\sharp \leq e$;
- for all $e \leq f$ in $E(\mathbf{S})$, $e^\sharp \leq f^\sharp$;
- if $\mathbf{S}$ is a monoid, $1^\sharp = 1$, we say then that $\mathbf{S}$ is a *stabilization monoid*

In this paper, we only consider finite stabilization semigroups. The intuition of the $\sharp$ operator is that $e^\sharp$ means "e repeated many times", which appears in the following properties, consequences of the definition above :

$$e^\sharp = e \cdot e^\sharp = e^\sharp \cdot e = e^\sharp \cdot e^\sharp = (e^\sharp)^\sharp \leq e$$

3.4. **Factorization trees and compatible function.** Let $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ be a stabilization semigroup, and $u \in S^*$. A *n-tree* t over u is a S-labelled tree such that u is the leaf word of t, and for each node p of t, we are in one of these case :

Leaf: p is a leaf,

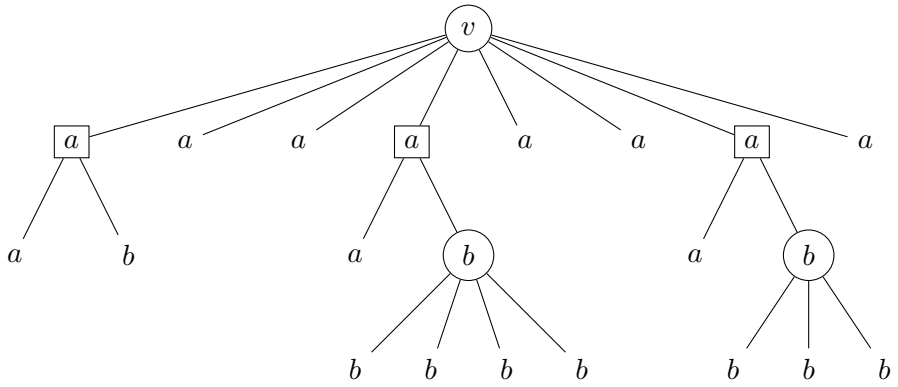
Binary : : p has only 2 children p_1, p_2, and $t(p) = t(p_1) \cdot t(p_2)$,

Idempotent : : p has k children $p_1, \ldots, p_k$ with $k \leq n$, and there is $e \in E(\mathbf{S})$ such that $t(p) = t(p_1) = \cdots = t(p_k) = e$,

Stabilization : : p has k children $p_1, \ldots, p_k$ with $k > n$, and there is $e \in E(\mathbf{S})$ such that $t(p_1) = \cdots = t(p_k) = e$, and $t(p) = e^\sharp$.

The root of t is called its *value* and is noted $\mathrm{val}(t)$.

Example 3.4. Let $u = abaaabbbbaaabbba$, $n \in \mathbb{N}$, and $S = \{a, b, \perp\}$ with $aa = ab = a$, $bb = b^\sharp = b$, and $a^\sharp = \perp$. The following tree is an n-tree over u :



Notice that the number of children of the root is $|u|_a = 8$. Two cases are possible :

- $n \leq 8$: the root is an idempotent node, and $v = a$.
- $n > 8$: the root is a stabilisation node, and $v = a^\sharp = \perp$.

This gives an intuition of how these factorization trees can be used to associate a value to a word, here its number of occurences of a.

In the following we will establish formally how we can use factorization trees to give a semantic to stabilization semigroups.

The following theorem is the cornerstone of this process. This theorem is a deep combinatoric result and generalizes Simon's factorization forests theorem. It can be considered as a Ramsey-like theorem, because it provides the existence of big well-behaved structures (the factorization tree, and in particular the idempotent nodes) if the input word is big enough.

Theorem 3.5. [Col09] *For all $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$, there exists $H \in \mathbb{N}$ such that for all $u \in S^*$ and $n \in \mathbb{N}$, there is a n-tree over u of height at most H.*

This allows us to define $\rho : S^+ \to \mathbb{N} \to S$ by

$$\rho(u)(n) = \min\{\mathrm{val}(t) : t \text{ is an } n\text{-tree over } u \text{ of height at most } H\}.$$

The function ρ is called *compatible* with $\mathbf{S}$. It depends on H so there may be several compatible functions, however we will see that they are equivalent in some sense.

If ρ is a function $S^+ \to \mathbb{N} \to S$, we associate to it a function $\tilde{\rho} : ((S^+)^{\mathbb{N}}) \to \mathbb{N} \to S$ by $\tilde{\rho}(\mathbf{u}) := \rho(\mathbf{u}(n))(n)$. We will also identify elements of $(S^{\mathbb{N}})^+$ with their canonic image in $(S^+)^{\mathbb{N}}$ (i.e. view a word of sequences as a sequence of words of same length).

Theorem 3.6. [Col09] *If ρ is a compatible function of $\mathbf{S}$, then there exists α such that :*
Letter.: *for all $a \in S, n \in \mathbb{N}$, $\rho(a)(n) = a$,*
Product.: *for all $a, b \in S$, $\rho(ab) \sim_\alpha a \cdot b$,*
Stabilization.: *for all $e \in E(\mathbf{S})$, $m \in \mathbb{N}$, $\rho(e^m) \sim_\alpha (e^\sharp|_m e)$,*
Substitution.: *for all $u_1, \dots, u_n \in S^+$, $n \in \mathbb{N}$, $\rho(u_1 \dots u_n) \sim_\alpha \tilde{\rho}(\rho(u_1) \dots \rho(u_n))$ (we identify sequence of words and word of sequences)*

Example 3.7. Let $\mathbf{S}$ be the stabilization semigroup with 3 elements $\bot \leq a \leq b$, with product defined by : $x \cdot y = \min_{\leq}(x, y)$ (b neutral element), and stabilization by $b^\sharp = b$ and $a^\sharp = \bot^\sharp = \bot$. Let $u \in \{\bot, a, b\}^+$, we define ρ by:

$$\rho(u) = \begin{cases} b & \text{if } u \in b^+ \\ \bot||u|_a a & \text{if } u \in b^*(ab^*)^+ \\ \bot & \text{otherwise.} \end{cases}$$

Then ρ is compatible with $\mathbf{S}$. This is proved by building a factorization tree of height 3, with idempotent (or stabilisation) b-nodes at level 3, binary nodes $a = a \cdot b$ at level 2, and one idempotent/stabilisation node at level 1.

3.5. **Recognized cost functions.** We now have all the mathematical tools to define how stabilization semigroups can recognize cost functions.

Let $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ be a stabilization semigroup. Let $h : \mathbb{A} \to S$ be a morphism, canonically extended to $h : \mathbb{A}^+ \to S^+$, and $I \subseteq S$ an ideal. let ρ be a compatible function associated with $\mathbf{S}, h$. We say that the quadruple $\mathbf{S}, h, I, \rho$ *recognizes* the function $f : \mathbb{A}^+ \to \mathbb{N}_\infty$ defined by

$$f(u) = I[\rho(h(u))] = \inf\{n \in \mathbb{N} : \rho(h(u))(n) \notin I\}.$$

We say that I is the *accepting ideal* of $\mathbf{S}$, it generalizes the accepting subset P used in the classical setting.

Indeed, if $\mathbf{S}, h, P$ is a classical semigroup recognizing $L \subseteq \mathbb{A}^+$ with an accepting subset P, we can take ρ to be the normal product $\pi : S^+ \to S$, $\sharp$ to be the identify function on idempotents, and I to be the complement of P. Then $\mathbf{S}, h, I, \pi$ recognizes χ_L.

Theorem 3.8. [Col09] *If ρ' satifies all the properties given in Theorem 3.6, then $\rho' \sim \rho$. In other words, ρ is unique up to $\sim$ (and in particular the choice of H is not important). Moreover, if $\mathbf{S}, h, I$ is given, and $\rho \sim \rho'$ are two compatible functions for $\mathbf{S}$, then the functions defined by $\mathbf{S}, h, I$ relatively to ρ and ρ' are equivalent up to $\approx$. This allows us to uniquely define the cost function $F = f^{\approx}$ recognized by the triplet $\mathbf{S}, h, I$, without ambiguity.*

Example 3.9. Let $\mathbb{A} = \{a, b\}$, the cost function $|\cdot|_a^{\approx}$ is recognizable. We take the stabilization semigroup from Example 3.7, h defined by $h(a) = a, h(b) = b$, and $I = \{\perp\}$. We have then $|u|_a = \inf\{n \in \mathbb{N} : \rho(h(u))(n) \neq \perp\}$ for all $u \in \mathbb{A}^+$.

The following theorem links cost automata with stabilization semigroups, and allows us to define the class of regular cost functions.

Theorem 3.10. [Col09] *Let F be a cost function, the following assertions are equivalent:*

- *F is recognized by a B-automaton,*
- *F is recognized by an S-automaton,*
- *F is recognized by a finite stabilization semigroup.*

Such a cost function F will be called regular *by generalization of this notion from language theory.*

Notice that if $L \subseteq \mathbb{A}^+$ is a language, then $\chi_L^{\approx}$ is a regular cost function if and only if L is a regular language. This shows that the notion of regularity for cost function is a proper extension of the one from language theory. That is to say, restricting cost functions theory to [regular] cost functions of the form $\chi_L^{\approx}$, one exactly gets [regular] language theory.

4. Quantitative LTL

We will now use an extension of LTL to describe some regular cost functions. This has been done successfully with regular languages, so we aim to obtain the same kind of results. Can we still go efficiently from an LTL-formula to an automaton?

4.1. **Definition.** The first thing to do is to extend LTL so that it can decribe cost functions instead of languages. We must add quantitative features, and this will be done by a new operator $U^{\leq N}$, required to appear positively in the formula. Unlike in most uses of LTL, we work here over finite words. This is to avoid additional technical considerations due to new formalisms suited to infinite words, which would make all the proofs heavier without adding any new ideas.

Formulas of $\text{LTL}^{\leq}$ (on finite words on an alphabet $\mathbb{A}$) are defined by the following grammar :

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi U^{\leq N} \varphi \mid \Omega$$

Where N is a unique free variable, common for all occurences of $U^{\leq N}$ operator. This is in the same spirit as in [KPV09], where the bound is global for all the formula.

- a means that the current letter is a, $\wedge$ and $\vee$ are the classical conjunction and disjunction;
- $X\varphi$ means that φ is true at the next letter;
- $\varphi U \psi$ means that ψ is true somewhere in the future, and φ holds until that point;
- $\varphi U^{\leq N} \psi$ means that ψ is true somewhere in the future, and φ can be false at most N times before ψ.
- Ω means that we are at the end of the word.

Notice the absence of negation in the syntax of $\text{LTL}^{\leq}$. However, we can still consider that $\text{LTL}^{\leq}$ generalizes classical LTL (with negation), because an LTL formula can be turned into an $\text{LTL}^{\leq}$ formula by pushing negations to the leaves. That is why we heed operators in dual forms in the syntax. Remark that we do not need a dual operator for U, because we can use Ω to negate it: $\neg(\varphi U \psi) \equiv \neg\psi U (\neg\varphi \vee \Omega)$. Moreover we can also express negations

of atomic letters: for all $a \in \mathbb{A}$ we can define $\neg a = (\bigvee_{b \neq a} b) \vee \Omega$ to signify that the current letter is not a.

We can then choose any particular $a \in \mathbb{A}$, and define $\top = a \vee \neg a$ and $\bot = a \wedge \neg a$, meaning respectively true and false.

We also define connectors "eventually" : $F\varphi = \top U \varphi$ and "globally" : $G\varphi = \varphi U \Omega$.

4.2. **Semantics.** We want to associate a function $[\![\varphi]\!]$ to any LTL$^{\leq}$-formula φ. As usual, we will often be more interested in the cost function $[\![\varphi]\!]^{\approx}$ recognized by φ.

We will say that $(u, n) \models \varphi$ (u, n is a model of φ) if φ is true on u with n as valuation for N, i.e. as number of errors for all the $U^{\leq N}$'s in the formula φ. We finally define

$$[\![\varphi]\!](u) = \inf \{n \in \mathbb{N} / (u, n) \models \varphi\}$$

We can remark that if $(u, n) \models \varphi$, then for all $k \geq n, (u, k) \models \varphi$, since the $U^{\leq N}$ operators appear always positively in the formula.

Proposition 4.1.

- $[\![a]\!](u) = 0$ *if* $u \in a\mathbb{A}^*$, *and* ∞ *otherwise*
- $[\![\Omega]\!](u) = 0$ *if* $u = \varepsilon$, *and* ∞ *otherwise*
- $[\![\varphi \wedge \psi]\!] = \max([\![\varphi]\!], [\![\psi]\!])$, *and* $[\![\varphi \vee \psi]\!] = \min([\![\varphi]\!], [\![\psi]\!])$
- $[\![X\varphi]\!](au) = [\![\varphi]\!](u)$, $[\![X\varphi]\!](\varepsilon) = \infty$
- $[\![\top]\!] = 0$, *and* $[\![\bot]\!] = \infty$

Example 4.2. Let $\varphi = (\neg a) U^{\leq N} \Omega$, then $[\![\varphi]\!] = |\cdot|_a$

We use LTL$^{\leq}$-formulae in order to describe cost functions, so we will often work modulo cost function equivalence $\approx$. However, we will sometimes be interested in the exact function $[\![\varphi]\!]$ described by φ.

Remark 4.3. If φ does not contain any operator $U^{\leq N}$, φ is a classical LTL-formula computing a language L, and $[\![\varphi]\!] = \chi_L$.

5. From LTL$^{\leq}$ to B-Automata

5.1. **Description of the automaton.** We will now give a direct translation from LTL$^{\leq}$-formulae to B-automata, i.e. given an LTL$^{\leq}$-formula ϕ on a finite alphabet $\mathbb{A}$, we want to build a B-automaton recognizing $[\![\phi]\!]^{\approx}$. We will also show that a slight change in the model of B-automaton (namely allowing sequences of counter actions on transitions) allows us to design a B-automaton $\mathcal{A}_\phi$ with $[\![\mathcal{A}_\phi]\!] = [\![\phi]\!]$: the functions recognized by $\mathcal{A}_\phi$ and ϕ are equal and not just equivalent up to $\approx$. This construction is adapted from the classic translation from LTL-formula to Büchi automata [DG10].

Let ϕ be an LTL$^{\leq}$-formula. We define $\mathrm{sub}(\phi)$ to be the set of subformulae of ϕ, and $Q = 2^{\mathrm{sub}(\phi)}$ to be the set of subsets of $\mathrm{sub}(\phi)$.

We want to define a B-automaton $\mathcal{A}_\phi = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$ such that $[\![\mathcal{A}]\!]_B \approx [\![\phi]\!]$.

We set the initial states to be $In = \{\{\phi\}\}$ and the final ones to be $Fin = \{\emptyset, \{\Omega\}\}$ We choose as set of counters $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ where k is the number of occurences of the $U^{\leq N}$ operators in ϕ, labeled from $U_1^{\leq N}$ to $U_k^{\leq N}$.

A state is basically the set of constraints we have to verify before the end of the word, so the only two accepting states are the one with no constraint, or with only constraint to be at the end of the word.

The following definitions are the same as for the classical case (LTL to Büchi automata):

Definition 5.1.

- An atomic formula is either a letter $a \in \mathbb{A}$ or Ω
- A set Z of formulae is consistent if there is at most one atomic formula in it.
- A reduced formula is either an atomic formula or a Next formula (of the form $X\varphi$).
- A set Z is reduced if all its elements are reduced formulae.
- If Z is consistent and reduced, we define $\text{next}(Z) = \{\varphi / X\varphi \in Z\}$.

Lemma 5.2 (Next Step)**.** *If Z is consistent and reduced, for all $u \in \mathbb{A}^*, a \in \mathbb{A}$ and $n \in \mathbb{N}$,*

$$(au, n) \models \bigwedge Z \text{ iff } (u, n) \models \bigwedge \text{next}(Z) \text{ and } Z \cup \{a\} \text{ consistent}$$

Proof. If $(au, n) \models \bigwedge Z$, then the only atomic formula that Z can contain is a, and therefore $Z \cup \{a\}$. Moreover, for every formula of the form $X\varphi$ in Z, we have $(au, n) \models X\varphi$. By definition of the semantic of the X operator, this means $(u, n) \models \varphi$. This is true for every $X\varphi$ in Z, so we obtain $(u, n) \models \text{next}(Z)$. The converse is similar. □

We would like to define $\mathcal{A}_\phi$ with $Z \longrightarrow \text{next}(Z)$ as transitions.

The problem is that $\text{next}(Z)$ is not consistent and reduced in general. If $\text{next}(Z)$ is inconsistent we remove it from the automaton. If it is consistent, we need to apply some reduction rules to get a reduced set of formulae. This consists in adding ε-transitions (but with possible actions on the counter) towards intermediate sets which are not actual states of the automaton (we will call them "pseudo-states"), until we reach a reduced set.

Let ψ be maximal (in size) not reduced in Y, we add the following transitions

- If $\psi = \varphi_1 \wedge \varphi_2$: $Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$
- If $\psi = \varphi_1 \vee \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- If $\psi = \varphi_1 U \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, X\psi\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- If $\psi = \varphi_1 U_j^{\leq N} \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, X\psi\} \\ Y \xrightarrow{\varepsilon:ic_j} Y \setminus \{\psi\} \cup \{X\psi\} \text{ (we count one mistake)} \\ Y \xrightarrow{\varepsilon:r_j} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$

where action r_j (resp. ic_j) perform r (resp. ic) on counter γ_j and ε on the other counters. The pseudo-states do not (a priori) belong to $Q = 2^{\text{sub}(\phi)}$ because we add formulae $X\psi$ for $\psi \in \text{sub}(\phi)$, so if Z is a reduced pseudo-state, $\text{next}(Z)$ will be in Q again since we remove the new next operators.

The transitions of automaton $\mathcal{A}_\phi$ will be defined as follows:

$$\Delta = \left\{ Y \xrightarrow{a:\sigma} \text{next}(Z) \mid Y \in Q, Z \cup \{a\} \text{ consistent and reduced}, Y \xrightarrow{\varepsilon:\sigma}_* Z \right\}$$

where $Y \xrightarrow{\varepsilon:\sigma}_* Z$ means that there is a sequence of ε-transitions from Y to Z with σ as combined action on counters.

5.2. **Correctness of $\mathcal{A}_\phi$.** We will now prove that $\mathcal{A}_\phi$ is correct, i.e. computes the same cost function as ϕ.

Definition 5.3. If σ is a sequence of actions on counters, we will call $\mathrm{val}(\sigma)$ the maximal value checked on a counter during σ with 0 as starting value of the counters, and $\mathrm{val}(\sigma) = 0$ if there is no check in σ. It corresponds to the value of a run of a B-automaton with σ as combined action of the counter.

Lemma 5.4. *Let $u = a_1 \dots a_m$ be a word on $\mathbb{A}$ and $Y_0 \overset{a_1:\sigma_1}{\to} Y_1 \overset{a_2:\sigma_2}{\to} \dots \overset{a_m:\sigma_m}{\to} Y_m$ an accepting run of $\mathcal{A}_\phi$.*

Then for all $\psi \in \mathrm{sub}(\phi)$, for all $n \in \{0, \dots, m\}$, for all $Y_n \overset{\varepsilon:\sigma}{\to}_* Y \overset{\varepsilon:\sigma'}{\to}_* Z$, verifying (if $n < m$) $Z \cup \{a_{n+1}\}$ consistent and reduced, and $Y_{n+1} = \mathrm{next}(Z)$

$$\psi \in Y \implies a_{n+1}a_{n+2} \dots a_m, N \models \psi$$

where $N = \mathrm{val}(\sigma'\sigma_{n+1} \dots \sigma_m)$.

Proof. We do a reverse induction on n.

If $n = m$, Y_n is a final state so $Y_n = \emptyset$ or $Y_n = \{\Omega\}$. If $Y_n \overset{\varepsilon:\sigma}{\to}_* Y$, then $Y = Y_n$ (no outgoing ε-transitions defined from $\emptyset$ or $\{\Omega\}$). Then if $\psi \in Y$, the only possibility is $\psi = \Omega$, but $a_{n+1} \dots a_m = \varepsilon$, and $\varepsilon, 0 \models \Omega$, hence the result is true for $n = m$.

Let $n < m$, we assume the result is true for $n+1$, and we take same notations as in the lemma, with $\psi \in Y$. By definition of Δ, there exists a transition $Y_n \overset{a_{n+1}:\sigma\sigma'}{\longrightarrow}_* \mathrm{next}(Z) = Y_{n+1}$ in $\mathcal{A}_\phi$.

We do an induction on the length k of the path $Y \overset{\varepsilon:\sigma'}{\to}_* Z$.

If $k = 0$, then $Y = Z$ is consistent and reduced, so ψ is either atomic or a Next formula.

If ψ is atomic, the only way $Z \cup \{a_{n+1}\}$ can be consistent is if $\psi = a_{n+1}$. In which case we obtain $a_{n+1} \dots a_m, N \models \psi$ without difficulty.

If $\psi = X\varphi$ with $\varphi \in \mathrm{next}(Z) = Y_{n+1}$, then it corresponds to the case $k = 0$. By induction hypothesis (on n), $a_{n+2} \dots a_m, N \models \varphi$ (N does not change because σ' is empty). Hence $a_{n+1}a_{n+2} \dots a_m, N \models X\varphi$ which shows the result.

If $k > 0$, we assume the result is true for $k-1$, and we show it for k. We have $Y \overset{\varepsilon:\sigma'_1}{\to} Y' \overset{\varepsilon:\sigma'_2}{\to}_* Z$ with $\sigma'_1\sigma'_2 = \sigma'$, and for all $\psi' \in Y', a_{n+1}a_{n+2} \dots a_m, N' \models \psi'$ with $N' = \mathrm{val}(\sigma'_2\sigma_{n+1} \dots \sigma_m)$.

We now look at the different possibilities for the ε-transition $Y \overset{\varepsilon:\sigma'_1}{\to} Y'$. Let us first notice that either $N = N'$ or $N = N' + 1$: since $\sigma'_1 \in \{\varepsilon, ic, r\}^\Gamma$, adding it at the beginning of a sequence can only increment its value by one, or leave it unchanged.

Let $u_{n+1} = a_{n+1}a_{n+2} \dots a_m$. If $\psi \in Y'$, then $u_{n+1}, N' \models \psi$, but $N \geq N'$ so $u_{n+1}, N' \models \psi$.

We just need to examine the cases where $\psi \notin Y'$:

- If $\psi = \varphi_1 \wedge \varphi_2$, $\sigma'_1 = \varepsilon$, and $Y' = Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$,
  then $u_{n+1}, N \models \varphi_1$ and $u_{n+1} \dots a_m, N \models \varphi_2$, hence $u_{n+1}, N \models \psi$.
- Other classical cases where $\sigma'_1 = \varepsilon$ are similar and come directly from the definition of LTL operators.
- If $\psi = \varphi_1 U_j^{\leq N} \varphi_2$, $\sigma'_1 = \varepsilon$ and $Y' = Y \setminus \{\psi\} \cup \{\varphi_1, X\psi\}$,
  then $u_{n+1}, N \models \varphi_1$ and $u_{n+1}, N \models X\psi$, hence $u_{n+1}, N \models \psi$
- If $\psi = \varphi_1 U_j^{\leq N} \varphi_2$, $\sigma'_1 = ic_j$ and $Y' = Y \setminus \{\psi\} \cup \{X\psi\}$,

then $u_{n+1}, N' \models X\psi$.

If γ_j reaches N' before its first reset in $\sigma'_2\sigma_{n+1}\dots\sigma_m$, then $N = N' + 1$, and we can conclude $u_{n+1}, N \models \psi$.

On the contrary, if $N = N'$ and there are strictly less than N' mistakes on φ_1 before the next occurence of φ_2, we can allow one more while still respecting the constraint with respect to $N = N' + 1$, so $u_{n+1}, N \models \psi$.

- If $\psi = \varphi_1 U_j^{\leq N} \varphi_2$, $\sigma'_1 = r_j$ and $Y' = Y \setminus \{\psi\} \cup \{\varphi_2\}$ then $N = N'$, and $u_{n+1}, N' \models X\varphi_2$, hence $u_{n+1}, N \models \psi$.

Hence we can conclude that for all k, $a_{n+1}a_{n+2}\dots a_m, N \models \psi$, which concludes the proof of the lemma. □

Lemma 5.4 implies the correctness of the automaton $\mathcal{A}_\phi$:
Let $Y_0 \overset{a_1:\sigma_1}{\to} Y_1 \overset{a_2:\sigma_2}{\to} \dots \overset{a_m:\sigma_m}{\to} Y_m$ be a valid run of $\mathcal{A}_\phi$ on u of value $N = [\![\mathcal{A}_\phi]\!]_B$, applying Lemma 5.4 with $n = 0$ and $Y = Y_0 = \{\phi\}$ gives us $(u, N) \models \phi$. Hence $[\![\phi]\!] \leq [\![\mathcal{A}_\phi]\!]_B$.

Conversely, let $N = [\![\phi]\!](u)$, then $(u, N) \models \phi$ so by definition of $\mathcal{A}_\phi$, it is straightforward to verify that there exists an accepting run of $\mathcal{A}_\phi$ over u of value $\leq N$ (each counter γ_i doing at most N mistakes relative to operator $U_i^{\leq N}$). Hence $[\![\mathcal{A}_\phi]\!]_B \leq [\![\phi]\!]$.

We finally get $[\![\mathcal{A}_\phi]\!]_B = [\![\phi]\!]$, the automaton $\mathcal{A}_\phi$ computes indeed the exact value of function $[\![\phi]\!]$ (and so we have obviously $[\![\mathcal{A}_\phi]\!]_B \approx [\![\phi]\!]$).

Contraction of actions. If we want to obtain a B-automaton as defined in Section 2.2, with atomic actions on transitions, we can proceed as follow.

We replace every action $\sigma \in \{\mathtt{ic}, \varepsilon, \mathtt{r}\}^*$ by the maximal letter atomic action $\max(\sigma)$ occuring in it, with respect to the order $\varepsilon < \mathtt{ic} < \mathtt{r}$. For instance $\mathtt{icricic}$ will be changed in $\mathtt{r}$. Let K be the maximal number of consecutive increments in such an action σ, that is to say

$$K = \max\{\mathrm{val}_B(\sigma) : (p, a, \sigma, q) \in \Delta\}.$$

Let $\mathcal{A}'$ be the automaton obtained from $\mathcal{A}_\phi$ by replacing each action σ by $\sigma' = \max(\sigma)$. Runs of $\mathcal{A}_\phi$ and $\mathcal{A}'$ are in a one-to-one correspondance in a canonic way: only counter actions have changed. Let ρ be a run of $\mathcal{A}_\phi$ and ρ' be the corresponding run of $\mathcal{A}'$.

First, remark that ρ' always perform less increments than ρ (going from σ to σ' only remove increments), so $\mathrm{val}_B(\rho') \leq \mathrm{val}_B(\rho)$.

Moreover, when we go from σ' to σ, we can add at most K increments before or after each reset, so between two resets of ρ (or edge of the word), we have at most $2K$ increments for each increment in ρ'. Let $\alpha(n) = 2Kn + 2K$, we obtain $\mathrm{val}_B(\rho) \leq \alpha(\mathrm{val}_B(\rho'))$.

Let $u \in \mathbb{A}^*$ and ρ a run of $\mathcal{A}_\phi$ such that $\mathrm{val}_B(\rho) = [\![\mathcal{A}_\phi]\!]_B(u)$. Then we saw that there a run ρ' of $\mathcal{A}'$ with $\mathrm{val}_B(\rho') \leq \mathrm{val}_B(\rho)$. Thus we obtain $[\![\mathcal{A}']\!]_B \leq [\![\mathcal{A}]\!]_B$.

Conversely, let ρ' be a run of $\mathcal{A}'$ such that $\mathrm{val}_B(\rho') = [\![\mathcal{A}']\!]_B(u)$. Then we saw that there is a run ρ of $\mathcal{A}$ with $\mathrm{val}_B(\rho) \leq_\alpha \mathrm{val}_B(\rho)$. Thus we obtain $[\![\mathcal{A}']\!]_B \preccurlyeq_\alpha [\![\mathcal{A}_\phi]\!]_B$.

In the end, we get $[\![\mathcal{A}']\!]_B \approx [\![\mathcal{A}_\phi]\!]_B = [\![\phi]\!]$, and $\mathcal{A}'$ is a B-automaton with atomic actions.

The results are summed up in the following theorem:

Theorem 5.5. *Let φ be an LTL$^{\leq}$-formula, we showed that $[\![\varphi]\!]^{\approx}$ is recognized by a B-automaton, and so $[\![\varphi]\!]^{\approx}$ is a regular cost function. If we authorize non-atomic actions on transitions, we can build a B-automaton that preserves the exact semantic of φ, not using approximation $\approx$.*

Since by [Col09], we can decide whether a function recognized by B-automaton is bounded, or even compare such functions with respect to $\preccurlyeq$, we get the following corollary:

Corollary 5.6. *Let φ and ψ be two LTL$^{\leq}$-formulae, we can decide whether $[\![\varphi]\!]$ is bounded, and more generally whether $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$ holds.*

Notice that deciding whether $[\![\varphi]\!]$ is bounded amounts to decide whether $[\![\varphi]\!] \preccurlyeq 0$ (where 0 is the function mapping every word to 0). Notice that boundedness of a formula corresponds to "uniform validity" of the formula: a formula is bounded if it can accepts every input, within a uniform bound N. In particular, a classical LTL formula is bounded if and only if it is true on all words. This is illustrated in Example 5.7.

Example 5.7. We give two examples on alphabet $\{a, b\}$: let $\varphi = (b \vee Xa \vee XFa)U^{\leq N}\Omega$, and $\psi = (a \vee Xa \vee XFa)U^{\leq N}\Omega$. Then $[\![\varphi]\!]$ is bounded by 2: the subformula $(b \vee Xa \vee XFa)$ fails if the remaining suffix is in ab^+ (which happens at most once), or if we are on the last letter and it is a. On the other hand, $[\![\psi]\!]$ is unbounded, because $[\![\psi]\!](b^n) = n$ for all n.

From [Col09], deciding whether $[\![\varphi]\!] \preccurlyeq [\![\psi]\!]$ requires to build an S-automaton recognizing $[\![\varphi]\!]^{\approx}$, and a B-automaton recognizing $[\![\psi]\!]^{\approx}$. This means that to test boundedness of a formula φ, we want to obtain an S-automaton recognizing $[\![\varphi]\!]^{\approx}$. Moreover, the standard algorithm translating between B- and S-automata is in EXPSPACE, because it uses the underlying stabilization semigroup, possibly containing exponentially many elements, compared to the number of states of automata.

To reduce the complexity of these two decision problems (boundedness and comparison of LTL$^{\leq}$-formulae), it is therefore useful to transform LTL$^{\leq}$-formulae directly into S-automata.

6. From LTL$^{\leq}$ to S-automata

In this section, we give a translation from LTL$^{\leq}$-formulae to the model of the S-automata. This will allow us to show that the boundedness problem for LTL$^{\leq}$-formulae is PSPACE-complete.

6.1. The logic LTL$^{>}$.

In order to naturally define a S-automaton from a LTL$^{\leq}$-formula, we will start be reversing the semantic of this formula.

Let LTL$^{>}$ be the logic defined by the following grammar:

$$\varphi := a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \ \varphi R^{>N} \varphi \mid \Omega$$

We want such a formula to be obtained by negating a LTL$^{\leq}$-formula, and then pushing negations to the leaves. This is why we need a dual operator to $U^{\leq N}$, which is $R^{>N}$. We want its semantic to be such that $(\neg\varphi)R^{>N}(\neg\psi)$ is equivalent to $\neg(\varphi U^{\leq N}\psi)$. That is to say, the semantic of $R^{>N}$ is defined by: $(u, n, i) \models \varphi R^{>N}\psi$ if for all $j > i$, either $(u, n, j) \models \psi$, or there are at least n positions $i \leq j' < j$ such that $(u, n, j') \models \varphi$. Other operators have same semantics as in LTL$^{\leq}$.

We can notice that if φ is a LTL$^{\leq}$-formula, then $\neg\varphi$ is equivalent to a LTL$^{>}$-formula, by pushing negations to the leaves.

If φ is a LTL$^{>}$-formula, we define the cost function $[\![\varphi]\!]_{\neg}$ recognized by φ by

$$[\![\varphi]\!]_{\neg}(u) = \sup\{n \in \mathbb{N} : (u, n) \models \varphi\}.$$

Lemma 6.1. *Let φ be a LTL$^{\leq}$-formula, then $[\![\neg\varphi]\!]_\neg \approx [\![\varphi]\!]$.*

Proof. Let φ be a LTL$^{\leq}$-formula, and $u \in \mathbb{A}^*$. If $[\![\varphi]\!](u) = \infty$, then for all $n \in \mathbb{N}$, $(u,n) \models \neg\varphi$, hence $[\![\neg\varphi]\!]_\neg = \infty$. Otherwise, let $n = [\![\varphi]\!](u) \in \mathbb{N}$, then $(u,n) \models \varphi$ and $(u,n+1) \not\models \varphi$. Thus we have $(u,n) \not\models \neg\varphi$ and $(u,n+1) \models \neg\varphi$, this implies $[\![\neg\varphi]\!]_\neg = n+1$.

This is enough to conclude $[\![\neg\varphi]\!]_\neg \approx [\![\varphi]\!]$. □

Going from an LTL$^{\leq}$-formula to its negation in LTL$^{\leq}$ can be done by a linear time algorithm: it suffices to push negations to the leaves, replacing each operator by the dual one (with possible addition of Ω). Thus it suffices to build the wanted S-automaton from an LTL$^{>}$-formula.

6.2. **From LTL$^{>}$ to S-automata.** Let ϕ be an LTL$^{>}$-formula, with k $R^{>N}$-operators, labelled $R_1^{>N}, R_2^{>N}, \dots, R_k^{>N}$.

We can build a S-automaton $\mathcal{A}_\phi$ as before, with counters $\{\gamma_1, \dots, \gamma_k\}$, by remembering subformulae of ϕ as constraints in states. The states of $\mathcal{A}_\phi$ are again $Q = 2^{\mathrm{sub}(\phi)}$, with $\{\phi\}$ as initial state. However, this time, the final states are every state Y such that for all $\varphi \in Y$, we have $\varphi = \Omega$ or φ is of the form $\varphi_1 R^{>N} \varphi_2$. Indeed, we have $(\epsilon, 0) \models \varphi_1 R^{>N} \varphi_2$, for any formulae φ_1 and φ_2. Then, the main new feature is how we deal with operators $R^{>N}$ in the table of ε-transitions between pseudo-states.

Let Y be a pseudo-state (or a real state), and ψ be a non-reduced formula of maximal size in Y. If ψ is not of the form $\varphi_1 R_j^{>N} \varphi_2$, then we add the same transitions as in Section 5.1.

Otherwise, if $\psi = \varphi_1 R_j^{>N} \varphi_2$, we add the following transitions:

$$\begin{cases} Y \xrightarrow{\varepsilon:\mathtt{i}_j} Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2, X\psi\} \text{ (we count one occurence of } \varphi_1\text{, and } \varphi_2 \text{ has to be seen)} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2, X\psi\} \text{ (we see } \varphi_2 \text{ without } \varphi_1\text{)} \\ Y \xrightarrow{\varepsilon:\mathtt{cr}_j} Y \setminus \{\psi\} \text{ (if } \varphi_2 \text{ cannot be proved, we perform } \mathtt{cr} \text{ to guarantee a lot of } \varphi_1 \text{ before)} \end{cases}$$

The proof of correctness is very similar to the one for B-automata in Section 5.2, so we omit it here. The main intuition is to that all transitions keep track of constraints in a sound way.

As before, we could build the transition table of $\mathcal{A}_\phi$ by contracting all ε-transitions, and verify that the resulting S-automaton recognizes $[\![\phi]\!]_\neg$. For our current purpose, it is not necessary to perform this contraction, we can think of $\mathcal{A}_\phi$ as having all the ε-transitions and pseudo-states described in its construction.

However, contracting sequences of S-actions will be useful in another context, to describe the PSPACE algorithm. So the next section describes how such sequences can be contracted.

6.3. **Semigroup of S-actions.** We will explicit how to contract S-actions, by using a stabilization semigroup $\mathbf{S}$ which contains all the necessary information about how to compose these actions. The product operation in $\mathbf{S}$ reflects the concatenation of S-actions. The stabilization operation $\sharp$ corresponds to repeating the same action a lot of times, for instance as it can be done in a cycle of the automaton. The element ω stands for a "big value", which can be made arbitrarily large, by repeating element $\mathtt{i}$ a lot of times. The element $\perp$ represents a fail of the run, when the automaton tries to perform action $\mathtt{cr}$ on a small counter value. The elements of $\mathbf{S}$ are gathered in the set $S = \{\omega, \mathtt{i}, \varepsilon, \mathtt{r}, \mathtt{cr}\omega, \mathtt{cr}, \perp\}$, and ordered by $\omega \leq \mathtt{i} \leq \varepsilon \leq (\mathtt{r}/\mathtt{cr}\omega) \leq \mathtt{cr} \leq \perp$. This order reflects a preference for the S-automaton:

between two actions $\sigma \leq \sigma'$, it is always better to choose σ in any context, when aiming for a big S-value. This explains why actions $\mathtt{r}$ and $\mathtt{cr}\omega$ are not comparable: the best choice can depend on the context. Indeed, in an empty context, action $\mathtt{r}$ is better (it yields value ∞ while $\mathtt{cr}\omega$ yields value 0), but in a context $C[x] = \omega \cdot x \cdot \mathtt{cr}$, it is better to choose $\mathtt{cr}\omega$ (yielding value ∞) than $\mathtt{r}$ (yielding value 0). Product and stabilization operations in $\mathbf{S}$ are explicited in the following array:

$\cdot$	ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	$^\sharp$
ω	ω	ω	ω	$\mathtt{r}$	ω	$\mathtt{r}$	$\bot$	ω
$\mathtt{i}$	ω	$\mathtt{i}$	$\mathtt{i}$	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	ω
ε	ω	$\mathtt{i}$	ε	$\mathtt{r}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	ε
$\mathtt{r}$	ω	$\mathtt{r}$	$\mathtt{r}$	$\mathtt{r}$	$\bot$	$\bot$	$\bot$	$\mathtt{r}$
$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\bot$	$\mathtt{cr}\omega$
$\mathtt{cr}$	$\mathtt{cr}\omega$	$\mathtt{cr}$	$\mathtt{cr}$	$\mathtt{cr}$	$\bot$	$\bot$	$\bot$	
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Notice that $\mathtt{cr}^\sharp$ is undefined because $\mathtt{cr}$ is not idempotent.

If Γ is a set of counters, then we denote by $\mathbf{S}^\Gamma$ the product stabilization semigroup with underlying set S^Γ, where all operations are performed component-wise. If $\sigma \in \mathbf{S}^\Gamma$ and $\gamma \in \Gamma$, we will note σ_γ the projection of σ on counter γ. When some components are not specified, the default value is ε. For instance if $\Gamma = \{1, 2\}$, we can write $\mathtt{i}_1$ for $(\mathtt{i}, \varepsilon)$ and $\mathtt{cr}\omega_2$ for $(\varepsilon, \mathtt{cr}\omega)$.

6.4. **Decision algorithm in polynomial space.** It was shown in [SC85] that satisfiability of classical LTL-formula is a PSPACE-complete problem. To obtain a PSPACE algorithm, an equivalent automaton is generated on-the-fly, and an accepting run of this automaton is guessed, while only information about the current state is remembered. Transition labels can be ignored, as only the existence of an accepting run is of interest, we do not care about which word can be accepted.

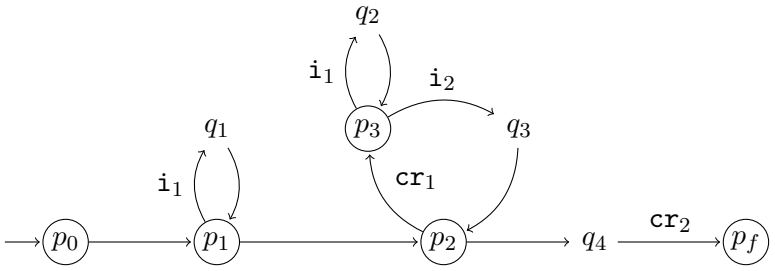
We want here to generalize this approach to cost functions: the problem is to explore the automaton $\mathcal{A}_\phi$ described earlier, but without keeping the whole automaton in the memory of the algorithm, in order to use only polynomial space with respect to the size of ϕ. We now want to decide whether the function $[\![\phi]\!]_\neg$ described by ϕ is bounded. This generalizes the satisfiability problem for LTL, since an LTL formula φ is satisfiable if and only if $[\![\neg\varphi]\!]$ is unbounded, that is to say $[\![\varphi]\!]_\neg$ is unbounded.

To do so, we will look for a witness of the fact that $[\![\mathcal{A}_\phi]\!]_S$ is unbounded. We have to face here an additional challenge compared to the classical case: it is not enough to find an accepting path, we have to find a family of accepting paths with arbitrary high values. This means that while we can forget the letters labelling transitions, we have to pay attention to counter actions.

We will need to keep information about counter values along the way. The principles that the algorithm has to respect for each counter $\gamma \in \Gamma$ are the following:

- Every action $\mathtt{cr}_\gamma$ must follow an action ω_γ, which represents a big number of increments $\mathtt{i}_\gamma$.
- The only way to obtain ω_γ is to go through a cycle containg at least one $\mathtt{i}_\gamma$, and only actions $\mathtt{i}$ and ε for γ.

Thus the aim is to describe a non-determinist algorithm that guesses a path in the automaton, as well as states that will be visited twice (in order to create cycles). These states will be called "control points". We explain how the algorithm works via the following example:



If $p_0 \in In$ and $p_f \in Fin$, the existence of this path in the automaton $\mathcal{A}_\phi$ is a witness that $[\![\mathcal{A}_\phi]\!]_S$ is unbounded. The aim of the algorithm is to find such a path, by guessing the beginning of each cycle, and contracting actions between two control points. In the example, the control points are p_1, p_2 and p_3. At any time, the memory of the algorithm contains a sequence $m, \sigma_1, p_1, \sigma_2, p_2, \ldots, \sigma_m, p_m$, where for all $i \in [1, m]$, action σ_i is in $\mathbf{S}^\Gamma$, and $p_i \in Q$. Moreover, p_m is always the current state of the run of $\mathcal{A}_\phi$, and $m \leq |\Gamma| + 1$. States $(p_i)_{i<m}$ are the current control points, i.e. starts of cycles that the run is currently using. They have to be closed in the future for the algorithm to end, and the last to be opened has to be the first closed, so that we get properly nested cycles. When a cycle is closed, operator $\sharp$ is applied to the action performed in the cycle, and then the product operation is used to concatenate this action with the one of the new innermost current cycle.

The aglorithm starts with memory $0, \varepsilon, p_0$ with $p_0 \in In$. In general p_0 can be chosen in a nondeterministic way, but here the unique initial state is $\{\phi\}$. Transitions are used on-the-fly: at any position, we guess a transition (available transitions depend only on formulae φ appearing in the current state), and we update the memory accordingly. The algorithm ends and outputs "unbounded" if the memory only contains $0, \sigma, p_f$, with $p_f \in Fin$, and for all $\gamma \in \Gamma$, $\sigma_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$.

We can remark that the condition $m \leq |\Gamma| + 1$ limits the number of nested cycles to $|\Gamma|$. This guarantees a memory space polynomial in $|\phi|$, since $|\Gamma| \leq |\phi|$ (every counter comes from an operator $R^{>N}$ of ϕ).

We come back to the above example, and we describe in the following table the successive statuses of the memory, while the algorithm guesses the wanted witness. For convenience, we focus here on the status of the memory when the algorithm passes through states q_1, q_2, q_3, q_4, p_f:

	m						
q_1	1	ε	p_1	$\mathtt{i}_1$	q_1		
q_2	2	ω_1	p_2	$\mathtt{cr}_1$	p_3	$\mathtt{i}_1$	q_2
q_3	1	ω_1	p_2	$(\mathtt{cr}\omega, \mathtt{i})$	q_3		
q_4	0	$(\mathtt{r}, \omega)$	q_4				
p_f	0	$(\mathtt{r}, \mathtt{r})$	p_f				

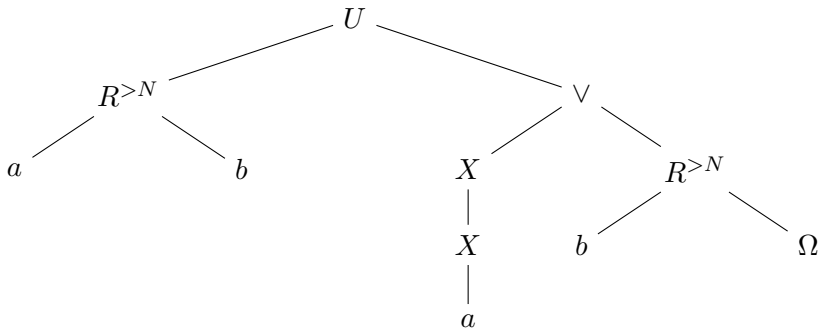
After passing through q_1, the algorithm goes back to p_1, and closes a cycle with only action $\mathtt{i}_1$. Thus it gets action $\mathtt{i}_1^\sharp = \omega_1$. Then the control points p_2 and p_3 are opened, with an action $\mathtt{cr}_1$ in-between. They are followed by an action $\mathtt{i}_1$, as we can see in state q_2. After state q_2, when the algorithm goes back to p_3, action $\mathtt{i}_2$ is stabilized, yielding ω_2. It

is then concatenated with the previous $\mathtt{cr}_1$ yielding action $(\mathtt{cr}\omega, \mathtt{i})$ that we can see in q_3. When the run goes back to p_2 and closes the external cycle, this $(\mathtt{cr}\omega, \mathtt{i})$ is stabilized into $(\mathtt{cr}\omega, \mathtt{i})^\sharp = (\mathtt{cr}\omega, \omega)$, and concatened with ω_1, yielding the $(\mathtt{r}, \omega)$ action that we can see in q_4. In the end, concatenation with the final $\mathtt{cr}_2$ yields global action $(\mathtt{r}, \mathtt{r})$ in p_f. By the acceptance condition of the algorithm, it can stop there and output "unbounded", since $\mathtt{r} \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$ and $p_f \in \mathit{Fin}$.

6.5. **Complexity and correctness of the algorithm.**

Lemma 6.2. *The algorithm described in Section 6.4 has a space complexity polynomial in* $|\phi|$.

Proof. We start by precising how the formula ϕ is given as input to the algorithm. Such a formula can be represented by a tree, whose nodes are operators, and whose leaves are atoms. For instance the formula $(aR^{>N}b)U((XXa) \vee (bR^{>N}\Omega))$ will be coded by the following tree:



This way, each subformula of ϕ corresponds to a node in this tree. A set of subformula is therefore just a set of nodes, and every state of the automaton can be encoded by a tuple $(n_1, n_2, \ldots, n_t)$, where every n_i encodes the position of a node of ϕ (it is easy to see that such an encoding is polynomial in the size of ϕ). Thus, the encoding of a state takes a polynomial space with respect to the size of the input tree, which is $|\phi|$.

We can remark that this is still true when adding pseudo-states, because those are subsets of $\mathrm{sub}(\phi) \cup \{X\varphi : \varphi \in \mathrm{sub}(\phi)\}$. Therefore, the encoding of a pseudo-state is at most twice as long as the encoding of a real state, so it still takes only polynomial space.

The encoding of an element in $|\mathbf{S}|$ takes constant space, so it takes a space linear in $|\Gamma|$ to encode an element of $\mathbf{S}^\Gamma$. Since every counter in $\mathcal{A}_\phi$ comes from an operator $R^{>N}$ of ϕ, we have $|\Gamma| \leq |\phi|$. Therefore, each element of $\mathbf{S}^\Gamma$ takes a space linear in $|\phi|$ in the memory.

Finally, we have $m \leq |\Gamma|$, so a space logarithmic in $|\phi|$ is enough to store m. At any time, we will have at most m pseudo-states and m elements of $\mathbf{S}^\Gamma$ in the memory, each one taking polynomial space in $|\phi|$. We can conclude that the whole sequence occupies a space which is only polynomial in $|\phi|$. □

Lemma 6.3. *The algorithm is correct, that is to say it outputs "unbounded" if and only if* $[\![\phi]\!]_\neg$ *is unbounded.*

Proof. It is easy to show that if the algorithm outputs "unbounded", then $[\![\phi]\!]_\neg$ is unbounded. Indeed, the algorithm describes a path (with cycles) in $\mathcal{A}_\phi$. It is straightforward to show that if every cycle is taken n times, the value of the resulting run is at least n. Therefore,

the path found by the algorithm describes a family of runs of arbitrarily high value, so we can conclude $[\![\phi]\!]_\neg$ is unbounded.

We now show the converse: we assume $[\![\mathcal{A}_\phi]\!]_S$ is unbounded, and we want to show that there exists a witness path that can be found by the algorithm. To do this, we define for all S-automaton $\mathcal{A}$ the stabilization semigroup $\mathbf{S}_\mathcal{A} = \langle S_\mathcal{A}, \cdot, \sharp, \leq \rangle$, whose elements represent sets of partial runs of $\mathcal{A}$. This construction parallels the one in [Col09].

A partial run from state p to state q performing global action σ will be represented by the element $(p, \sigma, q) \in Q \times S^\Gamma \times Q$.

If (p, σ, q) and (p', σ', q') are two elements of $Q \times S^\Gamma \times Q$, we will say that $(p, \sigma, q) \leq (p', \sigma', q')$ if $(p, q) = (p, q')$ and $\sigma \leq \sigma'$, for the ordered on S^Γ defined in Section 6.3.

Let $E \subseteq Q \times S^\Gamma \times Q$, we will denote by $E\downarrow = \{e \leq e' : e' \in E\}$ the downards-closure of E. Let $S_\mathcal{A} = 2^{Q \times S^\Gamma \times Q}\downarrow$ be the set of downwards-closed elements of $\mathbf{S}_\mathcal{A}$. Each element E of $S_\mathcal{A}$ represents a set of runs. The downwards-closure operation reflects the fact that we consider that the automaton is allowed to perform actions that are less efficient than the real ones: it does not change its global semantic.

Product and stabilization operation in $\mathbf{S}_\mathcal{A}$ are defined by:

$$E \cdot F = \{(p, act_1 \cdot \sigma_2, r) : (p, \sigma_1, q) \in E, (q, \sigma_2, r) \in F\}\downarrow$$
$$E^\sharp = \left\{(p, \sigma_1 \cdot \sigma_e^\sharp \cdot \sigma_2, r) : (p, \sigma_1, q), (q, \sigma_e, q), (q, \sigma_2, r) \in E\right\}\downarrow .$$

Notice that each element E describes a set of partial runs, and therefore, witnesses of accepting runs are described by the following subset of $S_\mathcal{A}$:

$$I' = \{E \in S_\mathcal{A} : \exists (p, \sigma, q) \in E, p \in In, q \in Fin, \forall \gamma \in \Gamma, \sigma_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}\},$$

together with the morphism $h : \mathbb{A} \to \mathbf{S}_\mathcal{A}$ by $h(a) = \{(p, \sigma, q) : (p, a, \sigma, q) \in \Delta_\mathcal{A}\}\downarrow$.

Since accepting ideals are defined as elements of big value, we take the accepting ideal to be $I = S_\mathcal{A} \setminus I'$.

It is not hard to verify that $S_\mathcal{A}, h, I$ recognizes the cost function $[\![\mathcal{A}]\!]_S$ (see [Col09] for more details). Consequently, $[\![\mathcal{A}]\!]_S$ is unbounded if and only if $\langle h(\mathbb{A})\rangle^\sharp \cap I \neq \emptyset$, i.e. there is an element of I that can be obtained from $h(\mathbb{A})$ via product and stabilization operations. Indeed, such an element can be described by a $\sharp$-expression e well-formed for $\mathbf{S}_\mathcal{A}$, with $\mathrm{eval}(e) \in I$, witnessing a sequence of words $(e(n))_{n\in\mathbb{N}}$ of unbounded value. We now apply this construction to the automaton $\mathcal{A}_\phi$ obtained from ϕ.

Since we assumed $[\![\mathcal{A}_\phi]\!]_S$ is unbounded, there exists a $\sharp$-expression e, well-formed for $\mathbf{S}_{\mathcal{A}_\phi}$, such that $\mathrm{eval}(e) \in I$. It remains to show that e does not need more than $|\Gamma|$ nested stabilization operators.

Let us assume that e contains at least $k = |\Gamma|+1$ nested stabilization operators $\sharp_1, \ldots, \sharp_k$, applied to $\sharp$-expressions $e_1, \ldots, e_k$ respectively. Let $(p_0, \sigma_f, p_f) \in \mathrm{eval}(e)$, witnessing the fact that $\mathrm{eval}(e) \in I$, that is to say $p_0 \in In, p_f \in Fin$, and for all $\gamma \in \Gamma, \sigma_\gamma \notin \{\mathtt{cr}, \mathtt{cr}\omega, \bot\}$. We will say that a stabilization operator $\sharp_i$ is *useful* if the element described by the $\sharp$-expression obtained from e by removing $\sharp_i$ does not contain (p_0, σ_f, p_f). We show by induction on $|\Gamma|$ that at least one of the operators $\sharp_1, \ldots, \sharp_k$ is not useful. If $|\Gamma| = 0$, then ϕ is a classic LTL formula, the automaton computes the characteristic function of a regular language, and stabilization is just the identity on $\mathbf{S}_{\mathcal{A}_\phi}$, so no stabilization operator can be useful. We now assume $|\Gamma| \geq 1$. Let $\sharp_k$ be the outmost stabilization operator in e. Therefore, we can write $e = x \cdot e_k^{\sharp_k} \cdot y$, where x and y are $\sharp$-expressions. let $E = \mathrm{eval}(e) = \mathrm{eval}(x)\mathrm{eval}(e_k^\sharp)\mathrm{eval}(y) = XE_k^\sharp Y$. We assume that $\sharp_k$ is useful (otherwise we get the wanted result). By definition

of the product of $\mathbf{S}_{\mathcal{A}_\phi}$, only one of the elements of $E_k^\sharp$ is used to obtain $(p_0, \sigma_f, p_f) \in E$. By definition of $\sharp$, this element is of the form (p, σ, r), with $(p, \sigma_1, q), (q, \sigma_e, q), (q, \sigma_2, r) \in E$ and $\sigma \leq \sigma_1 \cdot \sigma_e^\sharp \cdot \sigma_2$. Since $\sharp_k$ is useful, we must have $\sigma_e^\sharp \neq \sigma_e$, so there exists $\gamma \in \Gamma$ such that $(\sigma_e)_\gamma = \mathtt{i}$. Moreover, by definitions of the operations of $\mathbf{S}_{\mathcal{A}_\phi}$, the $\sharp$-expression e_k can not contain any useful stabilisation on counter γ. Therefore, we are left with the $k-1$ stabilisations in e_k, and $|\Gamma|-1$ available counters, since γ is no longer influenced by stabilizations. This concludes the proof by induction.

We can conclude that e is equivalent (with respect to eval) to an $\sharp$-expression e' with at most $|\Gamma|$ nested stabilizations. The fact that $\mathrm{eval}(e') \in I$ guarantees us the existence of a path in $\mathcal{A}_\phi$ that can be found by our algorithm, since it contains at most $|\Gamma|$ nested cycles. This concludes the proof of the Lemma. □

Theorem 6.4. *Given an LTL$^\leq$-formula ϕ, the problem of deciding whether $[\![\phi]\!]$ is bounded is PSPACE-complete.*

Proof. We saw that there exists a PSPACE algorithm solving this problem. We start be negating ϕ to obtain a formula ϕ' of LTL$^>$ (this is done in linear time). We then describe the transition table of the S-automaton $\mathcal{A}_{\phi'}$, and explore this automaton on-the-fly, using only polynomial space. This way we can guess a path witnessing unboundedness of $[\![\phi']\!]_\neg$, if such a path exists.

To show that the problem is PSPACE-hard, it suffices to remark that classical LTL satisfiability is a particular case of LTL$^\leq$ boundedness, and that LTL satisfiability is PSPACE-hard [SC85]. Indeed, if ϕ is a classical LTL-formula, we can see $\neg\phi$ as a formula of LTL$^\leq$, and we get that "$[\![\neg\phi]\!]$ bounded" is equivalent to "$L(\phi) = \emptyset$". □

We showed that generalisation of LTL into LTL$^\leq$ does not increase the computational complexity of the satisfiability/boundedness problem. This result is encouraging, since it allows us to treat a more general problem, without paying anything in terms of computational resources.

7. Syntactic congruence on $\omega\sharp$-expressions

We recall that as in the case of languages, stabilization semigroups recognize exactly regular cost functions, and there exists a quotient-wise minimal stabilization semigroup for each regular cost function [CKL10].

In standard theory, it is equivalent for a regular language to be described by an LTL-formula, or to be recognized by an aperiodic semigroup. Is it still the case in the framework of regular cost functions? To answer this question we first need to develop a little further the algebraic theory of regular cost functions.

7.1. Syntactic congruence.

In standard theory of languages, we can go from a description of a regular language L to a description of its syntactic monoid via the syntactic congruence. Moreover, when the language is not regular, we get an infinite monoid, so this equivalence can be used to "test" regularity of a language.

The main idea behind this equivalence is to identify words u and v if they "behave the same" relatively to the language L, i.e. L cannot separate u from v in any context : $\forall (x, y), xuy \in L \Leftrightarrow xvy \in L$.

The aim here is to define an analog to the syntactic congruence, but for regular cost functions instead of regular languages. Since cost functions look at quantitative aspects of words, the notions of "element" and "context" have to contain quantitative information : we want to be able to say things like "words with a lot of a's behave the same as words with a few a's".

That is why we will not define our equivalence over words, but over $\sharp$-expressions, which are a way to describe words with quantitative information.

7.2. **$\sharp$-expressions.** We first define general $\sharp$-expressions as in [Has90] and [CKL10] by just adding an operator $\sharp$ to words in order to repeat a subexpression "a lot of times". This differs from the stabilization monoid definition, in which the $\sharp$-operator can only be applied to specific elements (idempotents).

The set Expr of $\sharp$-expressions on an alphabet $\mathbb{A}$ is defined as follows:

$$e := a \in \mathbb{A} \mid ee \mid e^\sharp$$

If we choose a stabilization semigroup $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ together with a function $h : \mathbb{A} \to S$, the evaluation function eval : Expr $\to \mathbf{S}$ is defined inductively by $\mathrm{eval}(a) = h(a), \mathrm{eval}(ee') = \mathrm{eval}(e) \cdot \mathrm{eval}(e')$, and $\mathrm{eval}(e^\sharp) = \mathrm{eval}(e)^\sharp$ (eval(e) has to be idempotent). We say that e is *well-formed for* $\mathbf{S}$ if eval(e) exists. Intuitively, it means that $\sharp$ was applied to subexpressions that correspond to idempotent elements in $\mathbf{S}$.

If $f^{\approx}$ is a regular cost function, e is *well-formed for* f iff e is well-formed for the minimal stabilization semigroup of $f^{\approx}$.

Example 7.1. Let f be the function defined over $\{a\}^*$ by

$$f(a^n) = \begin{cases} n & \text{if } n \text{ even} \\ \infty & \text{otherwise} \end{cases}$$

The minimal stabilization semigroup of $f^{\approx}$ is : $\{a, aa, (aa)^\sharp, (aa)^\sharp a\}$, with $aa \cdot a = a$ and $(aa)^\sharp a \cdot a = (aa)^\sharp$. Hence the $\sharp$-expression $aaa(aa)^\sharp$ is well-formed for $f^{\approx}$ but the $\sharp$-expression $a^\sharp$ is not.

The $\sharp$-expressions that are not well-formed have to be removed from the set we want to quotient, in order to get only real elements of the syntactic semigroup.

7.3. **$\omega\sharp$-expressions.** We have defined the set of $\sharp$-expressions that we want to quotient to get the syntactic equivalence of cost functions. However, we saw that some of these $\sharp$-expressions may not evaluate properly relatively to the cost function $f^{\approx}$ we want to study, and therefore does not correspond to an element in the syntactic stabilization semigroup of $f^{\approx}$.

Thus we need to be careful about the stabilization operator, and apply it only to "idempotent $\sharp$-expressions". To reach this goal, we will add an "idempotent operator" ω on $\sharp$-expressions, which will always associate an idempotent element (relative to $f^{\approx}$) to a $\sharp$-expression, so that we can later apply $\sharp$ and be sure of creating well-formed expressions for f.

We define the set Oexpr of $\omega\sharp$-expressions on an alphabet $\mathbb{A}$:

$$E := a \in \mathbb{A} \mid EE \mid E^\omega \mid E^{\omega\sharp}$$

The intuition behind operator ω is that x^ω is the idempotent obtained by iterating x (which always exists in finite semigroups).

A *context* $C[x]$ is a $\omega\sharp$-expression with possible occurrences of a free variable x. Let E be a $\omega\sharp$-expression, $C[E]$ is the $\omega\sharp$-expression obtained by replacing all occurrences of x by E in $C[x]$, i.e. $C[E] = C[x][x \leftarrow E]$. Let $\mathrm{C_{OE}}$ be the set of contexts on $\omega\sharp$-expressions.

We will now formally define the semantic of operator ω, and use $\omega\sharp$-expressions to get a syntactic equivalence on cost functions, without mistyped $\sharp$-expressions.

Definition 7.2. If $E \in \mathrm{Oexpr}$ and $k, n \in \mathbb{N}$, we define $E(k,n)$ to be the word $E[\omega \leftarrow k, \sharp \leftarrow n]$, where exponentiation is relative to concatenation of words.

Lemma 7.3. *Let $F = f^{\approx}$ be a regular cost function, there exists $K_F \in \mathbb{N}$ such that for any $E \in \mathrm{Oexpr}$, the $\sharp$-expression $E[\omega \leftarrow K_F!]$ is well-formed for F, and we are in one of these two cases*

(1) $\forall k \geq K_F, \{f(E(k!,n)), n \in \mathbb{N}\}$ *is bounded : we say that* $E \in F^B$.
(2) $\forall k \geq K_F, \lim_{n\to\infty} f(E(k!,n)) = \infty$ *: we say that* $E \in F^\infty$.

Proof. Let $F = f^{\approx}$ be a regular cost function recognized by $\mathbf{S}_F, h, I$. Let $N = |\mathbf{S}_F|$. It suffices to take $K_F \geq N$ to verify that for any $E \in \mathrm{Oexpr}$, the $\sharp$-expression $E[\omega \leftarrow K_F!]$ is well-formed for F. Moreover, if $s \in \mathbf{S}_F$, $s^{k!} = s^{K_F!}$ for all $k \geq K_F$. Let us show that $F^\infty \uplus F^B = \mathrm{Oexpr}$. Let $E \in \mathrm{Oexpr}$, and $k \geq K_F$. Let $e = E[\omega \leftarrow k!]$, e is well-formed for $\mathbf{S}_F$. For all $n \in \mathbb{N}$, let $u_n = e(n) = E(k!, n)$. The structure of e directly gives us a factorization tree for u_n, the height of this tree depending only on e. Thus we know that there exists α (depending on e) such that $\rho(h(u_n)) \sim_\alpha \mathrm{eval}(e)|_n \mathrm{eval}(u_n)$.

Therefore,

$$\mathrm{eval}(e) \in I \Rightarrow \forall n, I[\rho(h(u_n))] \geq_\alpha n \Rightarrow \forall n, f(u_n) \geq_\alpha n \Rightarrow \lim f(u_n) = \infty$$

and $\mathrm{eval}(e) \notin I \Rightarrow \forall n, I[\rho(h(u_n))] \leq \alpha(1) \Rightarrow \forall n, f(u_n) \leq \alpha(1) \Rightarrow E \in F^B$. We get that $F^\infty = \{E \in \mathrm{Oexpr}, \mathrm{eval}(E) \in I\}$ and $F^B = \{E \in \mathrm{Oexpr}, \mathrm{eval}(E) \notin I\}$ which shows the result. □

Here, F^B and F^∞ are the analogs for regular cost functions of "being in L" and "not being in L" in language theory. But this notion is now asymptotic, since we look at boundedness properties of quantitative information on words. Moreover, F^∞ and F^B are only defined here for regular cost functions, since K_F might not exist if f is not regular.

Definition 7.4. Let F be a regular cost function, we write $E \rightleftharpoons_F E'$ if $(E \in F^B \Leftrightarrow E' \in F^B)$. Finally we define

$$E \equiv_F E' \text{ iff } \forall C[x] \in \mathrm{C_{OE}}, C[E] \rightleftharpoons_F C[E']$$

Remark 7.5. If $u, v \in \mathbb{A}^*$, and L is a regular language, then $u \sim_L v$ iff $u \equiv_{\chi_L} v$ ($\sim_L$ being the syntactic congruence of L). In this sense, $\equiv$ is an extension of the classic syntactic congruence on languages.

Now that we have properly defined the equivalence $\equiv_F$ over Oexpr, it remains to verify that it is indeed a good syntactic congruence, i.e. $\mathrm{Oexpr}/\equiv_F$ is the syntactic stabilization semigroup of F.

7.3.1. *Structure of* $\mathrm{Oexpr}/\equiv_F$. If F is a regular cost function, let $\mathbf{S}_F = \mathrm{Oexpr}/\equiv_F$. We show that we can provide $\mathbf{S}_F$ with a structure of stabilization semigroup $\langle \mathbf{S}_F, \cdot, \leq, \sharp \rangle$.

If $E \in \mathrm{Oexpr}$, let $\overline{E}$ be its equivalence class for the $\equiv_F$ relationship. We first naturally define the stabilization semigroup operators : $\overline{E} \cdot \overline{E'} = \overline{EE'}$ and if $\overline{E}$ idempotent we have $\overline{E} = \overline{E^\omega}$ and $(\overline{E})^\sharp = \overline{E^{\omega\sharp}}$. $\leq$ is the minimal partial order induced by the inequalities $s^\sharp \leq s$ where s is idempotent, and compatible with the stabilization semigroup structure.

Let us show that these operations are well-defined :

Product: If $E_1 \equiv_F E'_1$ and $E_2 \equiv_F E'_2$. By Lemma 7.7 with context xE_2 and $E'_1 x$, $E_1E_2 \equiv_F E'_1E_2 \equiv_F E'_1E'_2$, so $\overline{E_1E_2} = \overline{E'_1E'_2}$.

Stabilization: If $E \equiv_F E'$, by Lemma 7.7 with context $x^{\omega\sharp}$, $E^{\omega\sharp} \equiv_F E'^{\omega\sharp}$, hence $\overline{E^{\omega\sharp}} = \overline{E'^{\omega\sharp}}$.

Moreover, it is easy to check that all axioms of stabilization semigroups are verified, for example $(s^\sharp)^\sharp = s^\sharp$ because for any sequence u_n which is either bounded or tends towards ∞, u_{n^2} has same nature as u_n.

Theorem 7.6. $\mathbf{S}_F = \mathrm{Oexpr}/\equiv_F$ *is the minimal stabilization semigroup recognizing* f.

Proof. Let $I_F = \left\{\overline{E}, E \in F^\infty\right\}$, and $h_F : \mathbb{A}^* \to \mathbf{S}_F^*$ the length-preserving morphism defined by $h_F(a) = \overline{a}$ for all $a \in \mathbb{A}$ (a letter is a particular $\omega\sharp$-expression).

Let $\mathbf{S}_{\min}, h, I$ be the minimal stabilization semigroup recognizing F, as defined in appendix A.7 of [CKL10]. Let ρ be its compatible mapping, and eval : $\mathrm{Oexpr} \to \mathbf{S}_{\min}$ the corresponding evaluation function. We will show that $E \equiv_F E'$ iff $\mathrm{eval}(E) = \mathrm{eval}(E')$.

We know by the proof of Lemma 7.3 that $E \in F^\infty \Leftrightarrow \mathrm{eval}(E) \in I$. We recall that the definition of $\mathbf{S}_{\min}$ is based on the fact that if two elements behave the same relatively to I in any context, they are the same. These facts give us the following sequence of equivalences :

$$\begin{aligned} E \equiv_F E' &\Leftrightarrow \forall C[x] \in \mathrm{C}_{\mathrm{OE}}, C[E] \rightleftharpoons_F C[E'] \\ &\Leftrightarrow \forall C[x] \in \mathrm{C}_{\mathrm{OE}}, (C[E] \in F^\infty \Leftrightarrow C[E'] \in F^\infty) \\ &\Leftrightarrow \forall C[x] \in \mathrm{C}_{\mathrm{OE}}, (\mathrm{eval}(C[E]) \in I \Leftrightarrow (\mathrm{eval}(C[E']) \in I) \\ &\Leftrightarrow \mathrm{eval}(E) = \mathrm{eval}(E') \end{aligned}$$

This gives a bijection between $\mathbf{S}_F$ and $\mathbf{S}_{\min}$ (eval function is surjective on $\mathbf{S}_{\min}$, by minimality of $\mathbf{S}_{\min}$). Moreover, this bijection is an isomorphism, since in both semigroups, operations are induced by concatenation and $\sharp$ on $\sharp$-expressions. h is determined by its image on letters, so we have to define $h_F(a) = \overline{a}$ to remain coherent. Finally, we have $\mathrm{eval}(E) \in I \Leftrightarrow E \in F^\infty$, therefore the set I_F corresponding to I in the bijection is $I_F = \left\{\overline{E}, E \in F^\infty\right\}$. □

7.4. **Details on $\omega\sharp$-expressions.**

Lemma 7.7. *If* $E \equiv_F E'$, *then for any context* $C_1[x] \in \mathrm{C}_{\mathrm{OE}}$, $C_1[E] \equiv_F C_1[E']$.

Proof. Let E, E' and $C_1[x]$ defined by the Lemma. Let $C[x]$ be a context. We define $C'[x] = C[C_1[x]]$. The definition of the $\equiv_F$ relation implies $C'[E] \rightleftharpoons_F C'[e']$. Hence $C[C_1[e]] \rightleftharpoons_F C[C_1[E']]$.

This is true for any context $C[x]$ so $C_1[E] \equiv_F C_1[E']$. □

Proposition 7.8. *The relation $\equiv_F$ does not change if we restrict contexts to having only one occurence of x, as it was done for* Expr *in* [CKL10].

Proof. Let $\equiv'_F$ be the equivalence relation defined with single-variable contexts. we just need to show that $E \equiv'_F E' \implies E \equiv_F E'$ (the converse is trivial). Let us assume $E \equiv'_F E'$, and let $C[x_1, x_2]$ be a context with two occurences of x, labelled x_1 and x_2. Then $C[E] = C[x_1 \leftarrow E, x_2 \leftarrow E] \rightleftharpoons_F C[x_1 \leftarrow E, x_2 \leftarrow E'] \rightleftharpoons_F C[x_1 \leftarrow E', x_2 \leftarrow E'] = C[e']$. The generalization to an arbitrary number of occurences of x is obvious, and we get $E \equiv_F E'$. □

7.4.1. *Growing speeds lemma.* The following lemma will be used for technical purposes in future proofs. We state it here because it is an intuitive statement which can give a better understanding of the behaviour of regular cost functions and $\sharp$-expressions.

Lemma 7.9. *Let $F = f^{\approx}$ be a regular cost function, and $e \in$* Expr *containing N $\sharp$-operators $\sharp_1, \dots, \sharp_N$. For all $i \in \{1, \dots, N\}$, let σ_i be a function $\mathbb{N} \to \mathbb{N}$ with $\sigma_i(n) \to \infty$. Then*

$$f(e[\sharp_i \leftarrow \sigma_i(n) \text{ for all } i]) \to \infty \Leftrightarrow f(e(n)) \to \infty.$$

In other words, we can replace some n exponents by any function $\sigma(n) \to \infty$ when approximating a $\sharp$-expression by a sequence of words. It does not change the nature of the sequence relatively to f.

Proof. This result is intuitive : since we always work up to cost equivalence, growing at different speeds has an effect on correction functions, but not on qualitative behaviour.

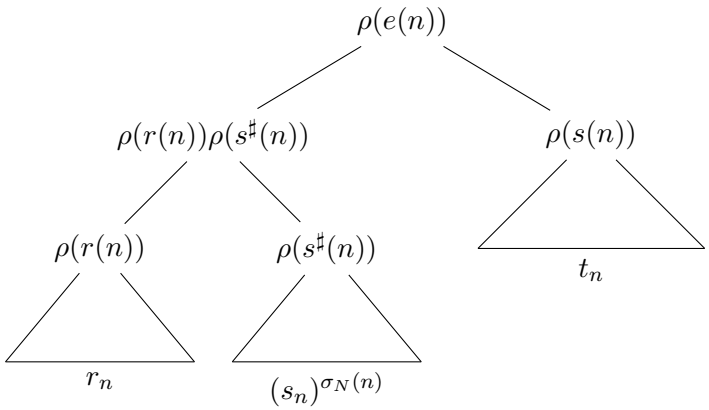
We will use notation $\underset{n\to\infty}{\bowtie}$: $g_1(n) \underset{n\to\infty}{\bowtie} g_2(n)$ means "$g_1(n)$ is bounded iff $g_2(n)$ is bounded". Remark that here all functions will either be bounded or tend towards ∞, thanks to the constraint that $\sharp$-expressions are well-formed for $\mathbf{S}_F$.

We will note $e_n = e[\sharp_i \leftarrow \sigma_i(n)$ pour tout $i]$. We want to show that $f(e_n) \underset{n\to\infty}{\bowtie} f(e(n))$. Let ρ be compatible with $\mathbf{S}_F$.

We want to show that there is α such that for all n, $\rho(e_n) \sim_\alpha \rho(e(n))$. We proceed by induction on N. If $N = 0$, then $e_n = e(n)$ and the result is trivial.

We suppose the result true for $k < N$, with function $\alpha_<$. Let $\sharp_N$ be an outmost stabilization operator (i.e. not nested in an other $\sharp$). We can write $e = rs^{\sharp_N}t$, with $r, s, t \in$ Expr, well-formed for $\mathbf{S}_F$, and eval$(s) \in E(\mathbf{S}_F)$.

By induction, there are n-trees of bounded height, and of value $\rho(r(n))$, $\rho(s(n))$ and $\rho(t(n))$ over r_n, s_n and t_n respectively. We can combine these trees by two binary nodes, and by a node which is either idempotent of stabilizing, in the following way:



The tree that we obtain can use sometimes n, sometimes $\sigma_N(n)$ as a threshold. It will be either an over-approximation or an under-approximation of the value of $f(n)$, with an error controlled by σ_N. Thus the sequence of values generated at the root is $\sim$-equivalent to $\rho(e(n))$, wihle the word e_n is always the leaf words. This concludes the proof of $\rho(e_n) \sim \rho(e(n))$ □

7.5. **Case of unregular cost functions.** The syntactic congruence can still be defined on unregular languages, and the number of equivalence classes becomes infinite, whereas a priori, we need cost functions to be regular to define their syntactic congruence.

Here, if $F = f^{\approx}$ is not regular, $\equiv_F$ may not be properly defined, since we use the existence of the minimal stabilization semigroup of F to give a semantic to the operator ω. But we can go back to $\sharp$-expressions and define $\sim_F$ on Expr for all f in the following way : $e \sim_F e'$ if for any context $C[x]$ on $\sharp$-expressions, the set $\{f(C[e])(n), n \in \mathbb{N}\}$ is bounded iff $\{f(C[e'])(n), n \in \mathbb{N}\}$ is bounded.

In this way if F is regular, then for all $e, e' \in$ Expr, $e \sim_F e'$ iff $e[\sharp \leftarrow \omega\sharp] \equiv_F e'[\sharp \leftarrow \omega\sharp]$. In particular Expr$/\sim_F$ is bigger than Oexpr$/\equiv_F$ when f is regular : there might be equivalence classes corresponding to $\sharp$-expressions that are not well-formed for F.

However, if F is not regular, Expr$/\sim_F$ is not infinite in general (this differs from the results in language theory).

Example 7.10. Let $f(u) = \min_{e \in \text{Expr}} \{|e|, \exists n \in \mathbb{N}, u = e(n)\}$, and $F = f^{\approx}$, there is only one equivalence class for $\sim_F$, because $f(C[e](n))$ is always bounded by $|C[e]|$. So Expr$/\sim_F$ has only one element, and therefore cannot contain a stabilization semigroup computing F. This gives us a proof that F is not regular.

8. Expressive power of LTL$^{\leq}$

If F is a regular cost function, we will call $\mathbf{S}_F$ the syntactic stabilization semigroup of F.

A finite semigroup $\mathbf{S} = \langle S, \cdot \rangle$ is called *aperiodic* if $\exists k \in \mathbb{N}, \forall s \in \mathbf{S}, s^{k+1} = s^k$. The definition is the same if $\mathbf{S}$ is a finite stabilization semigroup.

Remark 8.1. For a regular cost function F, the statements "F is recognized by an aperiodic stabilization semigroup" and "$\mathbf{S}_F$ is aperiodic" are equivalent, since $\mathbf{S}_F$ is a quotient of all stabilization semigroups recognizing F.

8.1. From LTL$^{\leq}$ to Aperiodic Stabilization Semigroups.

Theorem 8.2. *Let F be a cost function described by a LTL$^{\leq}$-formula, then F is regular and the syntactic stabilization semigroup of F is aperiodic.*

The proof of this theorem will be the first framework to use the syntactic congruence on cost functions.

Proof. We want to show that for all LTL$^{\leq}$-formula ϕ, $\mathbf{S}_{[\![\phi]\!]^{\approx}}$ is aperiodic.

We proceed by an induction on ϕ and use the characterization of $\mathbf{S}_{[\![\phi]\!]^{\approx}}$ provided by Theorem 7.6.

8.1.1. *Case $\phi = a$.*

We have $S_{[\![\phi]\!]^{\approx}} = \{a, b\}$ with $a \cdot b = a \cdot a = a$, and $b \cdot a = b \cdot b = b$, it is aperiodic (also trivial if $\phi = \neg a$).

8.1.2. *Case $\phi = \Omega$.*

Then $S_{[\![\phi]\!]^{\approx}} = \{1, a\}$ with 1 neutral element and $a \cdot a = a$, it is aperiodic.

8.1.3. *Case $\phi = \varphi_1 \wedge \varphi_2$ or $\phi = \varphi_1 \vee \varphi_2$.*

ϕ is recognized by the product semigroup of $\mathbf{S}_{[\![\varphi_1]\!]}$ and $\mathbf{S}_{[\![\varphi_2]\!]}$, which is aperiodic by induction hypothesis.

8.1.4. *Case $\phi = X\psi$.*

We know by induction hypothesis that $\mathbf{S}_{[\![\psi]\!]^{\approx}}$ is aperiodic, so there exists $k \in \mathbb{N}$ such that for any $\omega\sharp$-expression E, $E^k \equiv_{[\![\psi]\!]} E^{k+1}$. We want to show that it is also true for $[\![\phi]\!]$. Let E be a $\omega\sharp$-expression, and $e = E[\omega \leftarrow \max(K_{[\![\phi]\!]^{\approx}}!, K_{[\![\psi]\!]^{\approx}}!)]$ (from Lemma 7.3).

We want to show that $E^{k+2} \equiv_{[\![\phi]\!]^{\approx}} E^{k+1}$ i.e. for any context $C[x]$,

$$[\![\phi]\!](C[e^{k+2}](n)) \underset{n\to\infty}{\bowtie} [\![\phi]\!](C[e^{k+1}](n)).$$

Let $C[x]$ be a context.

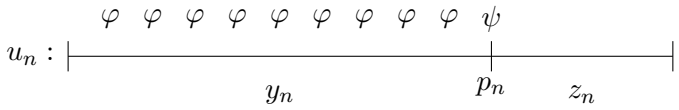
- If $C[x] = aC'[x]$, then by proposition 7.7 with context xe:
$$[\![\phi]\!](C[e^{k+2}](n)) = [\![\psi]\!](C'[e^{k+2}](n)) \underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^{k+1}](n)) = [\![\phi]\!](C[e^{k+1}](n)).$$
- If the beginning of $C[x]$ is a letter a under (at least) a $\sharp$, we have a context $C'[x]$ such that for any $\sharp$-expression e', $C[e'](n+1) = aC'[e'](n)$. For instance if $C[x] = ((ax)^\sharp b)^\sharp$ then $C'[x] = x(ax)^\sharp b((ax)^\sharp b)^\sharp$. Then we can write $[\![\phi]\!](C[e^{k+2}](n+1)) = [\![\psi]\!](C'[e^{k+2}](n)) \underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^{k+1}](n)) = [\![\phi]\!](C[e^{k+1}](n+1))$.
- Finally, if $C[x]$ starts with x (possibly under $\sharp$), we expand x in ex in $C[x]$, so that it does not start with x anymore. As before we can get $C'[x]$ such that $C[e^{k+1}](n+1) = aC'[e^k](n)$ and $C[e^{k+2}](n+1) = aC'[e^{k+1}](n)$ for all n, hence

$$\begin{aligned}
[\![\phi]\!](C[e^{k+2}](n+1)) &= [\![\phi]\!](aC'[e^{k+1}](n)) \\
&= [\![\psi]\!](C'[e^{k+1}](n)) \\
&\underset{n\to\infty}{\bowtie} [\![\psi]\!](C'[e^k](n)) \\
&= [\![\phi]\!](aC'[e^k](n)) \\
&= [\![\phi]\!](C[e^{k+1}](n+1))
\end{aligned}$$

8.1.5. *Case* $\phi = \varphi U \psi$.

we know by induction hypothesis that $\mathbf{S}_{[\![\varphi]\!]^\approx}$ and $\mathbf{S}_{[\![\psi]\!]^\approx}$ are aperiodic, so there exists $k \in \mathbb{N}$ such that for any $\omega\sharp$-expression E, $E^k \equiv_{[\![\varphi]\!]} E^{k+1}$ and $E^k \equiv_{[\![\psi\approx]\!]} E^{k+1}$. Let E be a $\omega\sharp$-expression. We will show that $E^{k+1} \equiv_{[\![\phi]\!]^\approx} E^{k+2}$

Let $C[x]$ be a context in $\mathrm{C_{OE}}$, $K = \max(K_{[\![\varphi]\!]^\approx}, K_{[\![\psi]\!]^\approx})$, $u_n = C[E^{k+1}](K!, n)$ and $v_n = C[E^{k+2}](K!, n)$. We want to show that $C[E^{k+1}] \rightleftharpoons_{[\![\phi]\!]^\approx} C[E^{k+2}]$, i.e. $[\![\phi]\!](u_n) \underset{n\to\infty}{\bowtie} [\![\phi]\!](v_n)$. Assume for example that $[\![\phi]\!](u_n)$ is bounded by m We have $u_n, m \models \phi$ for all n. We can write $u_n = y_n z_n$ with $z_n, m \models \psi$ and for any strict suffix y_n^i of y_n, $y_n^i z_n, m \models \varphi$. Let p_n be the starting position of z_n (position 0 being the beginning of the word). We define y_n^i to be the suffix of y_n starting at position i for all $i \in [\![0, p-1]\!]$. In this way $y_n^0 = y_n$.



Let us focus on the position p_n of the beginning on z_n. The $\sharp$-expression $e = C[E^{k+1}](K!)$ is finite so we can extract a sequence $u_{\delta(n)}$ from u_n such that the beginning position $p_{\delta(n)}$ of $z_{\delta(n)}$ corresponds to the same position p in e. Let $\{e_j, j \in J\}$ be the finite set of $\sharp$-expression such that $e_j^\sharp$ contains position p in e. We choose $J = \{1, r\}$ with $1 \leq j < j' \leq r$ implies $e_j^\sharp$ is a subexpression of $e_{j'}$. For convenience, we label the $\sharp$-operator of $e_j^\sharp$ with j. Note that J can be empty, if p does not occur under a $\sharp$ in e.

We denote by $\overleftarrow{f_j}(\delta(n))$ the number of occurences of $e_j(\delta(n))$ (coming from the corresponding $e_j^\sharp$) in $y_{\delta(n)}$ and we define $\overrightarrow{f_j}(\delta(n))$ in the same way relatively to $z_{\delta(n)}$. We have for all $n \in \mathbb{N}$, $\delta(n) - 1 \leq \overleftarrow{f_j}(\delta(n)) + \overrightarrow{f_j}(\delta(n)) \leq \delta(n)$. The $\delta(n) - 1$ lower bound is due to the fact than p can be in the middle of one occurence of e_j, therefore this occurence does not appear in $y_{\delta(n)}$ nor in $z_{\delta(n)}$.

This implies that for each $j \in J$, we are in one of these three cases:

- $j \in J_1$: $\overleftarrow{f_j}(\delta(n))$ is unbounded and $\overrightarrow{f_j}(\delta(n))$ is bounded.
- $j \in J_2$: $\overleftarrow{f_j}(\delta(n))$ is bounded and $\overrightarrow{f_j}(\delta(n))$ is unbounded.
- $j \in J_3$: $\overleftarrow{f_j}(\delta(n))$ and $\overrightarrow{f_j}(\delta(n))$ are both unbounded .

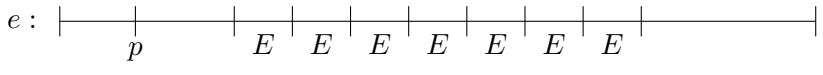
But J is finite, hence we can extract $\sigma(n)$ from $\delta(n)$ such that for each $j \in J$:

- If $j \in J_1$, $\overleftarrow{f_j}(\sigma(n)) \to \infty$ and $\overrightarrow{f_j}(\sigma(n))$ is constant.
- If $j \in J_2$, $\overleftarrow{f_j}(\sigma(n))$ is constant and $\overrightarrow{f_j}(\sigma(n)) \to \infty$.
- If $j \in J_3$, $\overleftarrow{f_j}(\sigma(n)) \to \infty$ and $\overrightarrow{f_j}(\sigma(n)) \to \infty$.

Remark that if $j < j'$ and $\overrightarrow{f_j} \circ \sigma \neq 0$, then $j \notin J_1$. Symmetrically, if $j < j'$ and $\overleftarrow{f_j} \circ \sigma \neq 0$, then $j \notin J_2$.

We can distinguish three cases for the position of p in $e = C[E^{k+1}](K!)$:

First case: p is before the first occurence of E in e.



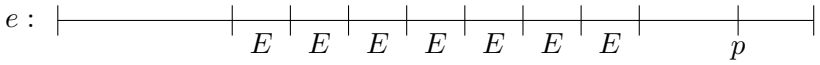
We consider $C'[x] \in \mathrm{C_{OE}}$ obtained from $C[x]$ by replacing $\sharp_j$ by the constant value of $\overrightarrow{f_j}(\sigma(n))$ for all $j \in J_1$. We have $[\![\psi]\!](z_{\sigma(n)}) \leq m$ for all n, but by Lemma 7.9, $[\![\psi]\!](z_n)$ is bounded iff $C'[E^{k+1}] \in ([\![\psi]\!]^{\approx})^B$. By induction hypothesis, $C'[E^{k+1}] \in [\![\psi]\!]^B \Leftrightarrow C'[E^{k+2}]] \in ([\![\psi]\!]^{\approx})^B$. Let z'_n be the suffix of $C[E^{k+2}](K!, n)$ starting at position p_n. By reusing Lemma 7.9, we get that $[\![\psi]\!](z'_{\sigma(n)}) \leq m'$ for some m'.

We still have to show that there exists a constant M such that $[\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)}) \leq M$ for all n and all $i \in [\![1, p_{\sigma(n)}]\!]$ (the $y^i_{\sigma(n)}$ are not affected by the change from E^{k+1} to E^{k+2}). Let us call $g^i_{\sigma(n)} = [\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)})$ for more lisibility. Let us assume that no such M exists, then $\left\{ g^i_{\sigma(n)} : n \in \mathbb{N}, 1 \leq i \leq p_{\sigma(n)} \right\}$ is unbounded. For all n, we define i_n such that $g^{i_{\sigma(n)}}_{\sigma(n)} = \max \left\{ g^i_{\sigma(n)} : 1 \leq i \leq p_{\sigma(n)} \right\}$. By construction, the sequence $g^{i_{\sigma(n)}}_{\sigma(n)} = [\![\varphi]\!](y^{i_{\sigma(n)}}_{\sigma(n)} z'_{\sigma(n)})$ is unbounded. We first extract $\sigma'(n)$ from $\sigma(n)$ such that $g^{i_{\sigma'(n)}}_{\sigma'(n)} \to \infty$.

We can now repeat the same process as before to extract a sequence $\gamma(n)$ from $\sigma'(n)$, such that the starting positions of $y^{i_{\gamma(n)}}_{\gamma(n)}$ for all n correspond to the same position in e, and such that there exists a context $C''[x]$ with $[\![\varphi]\!](y^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)}) \underset{n\to\infty}{\bowtie} [\![\varphi]\!](C''[E^{k+1}](K!, \gamma(n)))$ (by Lemma 7.9 again). By adding an extra E (from $k+1$ to $k+2$) and changing z by z' (the y factors are not concerned by occurences of E), we get $g^{i_{\gamma(n)}}_{\gamma(n)} \underset{n\to\infty}{\bowtie} [\![\varphi]\!](C''[E^{k+2}](K!, \gamma(n)))$. By hypothesis, $[\![\varphi]\!](y^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)})$ bounded by m, and $C''[E^{k+1}] \rightleftharpoons_{[\![\varphi]\!]} C''[E^{k+2}]$, so $g^{i_{\gamma(n)}}_{\gamma(n)}$ is bounded, but we already know that $g^{i_{\gamma(n)}}_{\gamma(n)} \to \infty$. We have a contradiction, so M must exist.

We finally obtain the existence of M such that for all n and valid i, $[\![\varphi]\!](y^i_{\sigma(n)} z'_{\sigma(n)}) \leq M$. This together with the previous result on ψ gives us that $[\![\varphi U \psi]\!](C[E^{k+2}](K!, n)) \leq \max(m', M)$. We got $C[E^{k+1}] \in [\![\phi]\!]^B \implies C[E^{k+2}] \in [\![\phi]\!]^{\approx B}$. The other direction works exactly the same, by removing one E instead of adding one. Hence we have $C[E^{k+1}] \rightleftharpoons_{[\![\phi]\!]^{\approx}} C[E^{k+2}]$.

Second case: p is after the last occurence of E in e.



This time z_n is not affected by changing from E^{k+1} to E^{k+2}, however it affects some of the y^i_n. Let $y'^i_n z_n$ be the suffixes of $v_n = C[E^{k+2}](K!, n)$, and p'_n the position of the beginning of z_n in v_n.

As before, we assume that $\left\{ [\![\varphi]\!](y'^i_{\sigma(n)} z_{\sigma(n)}) : n \in \mathbb{N}, 1 \leq i \leq p_{\sigma(n)} \right\}$ is unbounded, and we build a sequence $y'^{i_{\gamma(n)}}_{\gamma(n)}$ with the same start position in e, such that $[\![\varphi]\!](y'^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)}) \to \infty$.

We can again extract context $C''[x]$, but we may need to use again Lemma 7.9, in order to map the $\sharp$'s of $C''[x]$ with the remaining repetitions of idempotent elements, (which could be any functions $g(n) < n$). The main idea is to map positions in $v_{\gamma(n)}$ with positions in $u_{\gamma(n)}$ in order to be able to bound the values $[\![\varphi]\!](y'^{i_{\gamma(n)}}_{\gamma(n)} z_{\gamma(n)})$ with what we know about the behaviour on $u_{\gamma(n)}$, and so get a contradiction. Three cases are to be distinguished:

- If a factor corresponding to E^{k+2} occurs in the $y'^{i_{\gamma(n)}}_{\gamma(n)}$, the precedent proof stays valid, and we can map $y'^{i_{\gamma(n)}}_{\gamma(n)}$ with some $y^{j_{\gamma(n)}}_{\gamma(n)}$ ($j_{\gamma(n)}$ may be different from $i_{\gamma(n)}$) in order to get the contradiction. The mapping just need to take in account the shift due to the new occurences of E, but the positions in the words are essentially the sames.
- If the remaining factors contain at most k occurences of E, then the position can be matched with positions in u_n without any changes, and we get the contradiction.
- If the remaining factors contain $k+1$ occurences of E, then we can use the equivalence $E^{k+1} \equiv_{[\![\varphi]\!]\approx} E^k$ to match positions in v_n with positions in u_n and get the contradiction. This time we map positions in the first E of each sequence E^k with the corresponding position in the second one. Informally, we "duplicate" the first E of each sequence.

To sum up, the following figure shows how positions of v_n are mapped with positions of u_n, in the case $k = 2$. This figure is just an example, and is simpler than the general case, because only one sequence of E's appear here. If an other sequence appears before, all the positions are shifted, but the general principle stays the same.

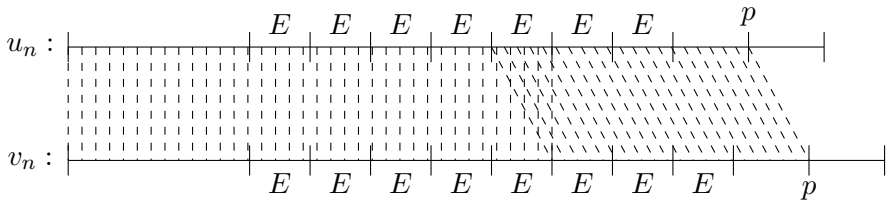
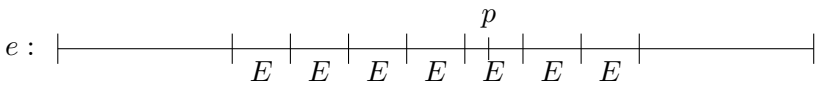


Figure 1: Association of positions

This choice of association is arbitrary: one can indeed choose any E to duplicate, we will still be able to use the induction hypothesis on u_n, or the hypothesis that φ is true on suffixes of u_n starting before p, in order to conclude.

Third case:



In all other situations, a combination of the techniques used above gives us the wanted result. We just need to do with ψ what we did with φ in the second case: for instance we may use $E^{k+1} \equiv_{[\![\psi]\!]\approx} E^k$ if $z'_{\sigma(n)}$ contains $k+1$ occurences of E.

As before, the other way is similar, and we finally get $E^{k+1} \equiv_{[\![\phi]\!]\approx} E^{k+2}$. In conclusion, $\mathbf{S}_{[\![\phi]\!]\approx}$ is aperiodic.

8.1.6. *Case* $\phi = \varphi U^{\leq N} \psi$.

We just need to adapt the precedent proof to take in account some exceptions in the validities of φ formulae. Indeed removing an occurence of E does not change the number of possible mistakes, but adding one can double it (at worse), since at most two positions in v_n are mapped to the same position in u_n. Hence , under the hypotheses $E^k \equiv_{[\![\psi]\!]^\approx} E^{k+1}$ and $E^k \equiv_{[\![\varphi]\!]^\approx} E^{k+1}$, we get $E^{k+1} \equiv_{[\![\varphi U^{\leq N} \psi]\!]^\approx} E^{k+2}$, with a correction function that doubles the one in the precedent proof. We can conclude that $\mathbf{S}_{[\![\phi]\!]^\approx}$ is also aperiodic in this case. □

8.2. From Aperiodic Stabilization Semigroups to LTL$^{\leq}$.

Theorem 8.3. *Let F be a cost function recognized by an aperiodic stabilization semigroup, then F can be described by an LTL$^{\leq}$-formula.*

Proof. This proof is a generalization of the proof from Wilke for aperiodic languages in [Wil99].

Let us first notice that "$\mathbf{S}_F$ is aperiodic" is equivalent to "F is computed by an aperiodic stabilization monoid", since aperiodicity is preserved by quotient and by addition of a neutral element.

We take an alphabet $\mathbb{A} \subseteq \mathbf{M}$ to avoid using a morphism h and simplify the proof. The LTL$^{\leq}$-formulae are about elements of $\mathbf{M}$, and are monotonic in the sense that $[\![a]\!](bu) = 0$ iff $b \geq a$, ∞ otherwise. It is easy to get from this to the general case by substituting in the formula an element m by $\vee_{h(a) \geq m} a$. We also will be sloppy with the empty word ε. It is not more difficult to take it in account, but the addition of a lot of special cases for ε in the proof would make it harder to understand.

We assume that $F = f^{\approx}$ on alphabet $\mathbb{A} \subseteq \mathbf{M}$ is computed by $\mathbf{M}, idI$ with $\mathbf{M}$ aperiodic. Let ρ be compatible with $\mathbf{M}$.

If $m \in \mathbf{M}$, we note f_m the cost function $f_m(u) = \inf\{n : \rho(u)(n) \geq m\}$. It is sufficient to show that the f_m functions are LTL$^{\leq}$-computable up to $\approx$, since $f \approx \min_{m \notin I} f_m$.

We proceed by induction on both the size of the stabilization monoid and on the size of the alphabet, the induction parameter being $(|\mathbf{M}|, |\mathbb{A}|)$ for order $\leq_{lex}$.

We add in the induction hypothesis that $\mathbf{M}$ has a neutral element 1 for multiplication.

If $|\mathbf{M}| = 1$, i.e. $M = \{a\}$, then f_a is the constant function 0 or ∞, which is LTL$^{\leq}$-computable.

If $\mathbb{A} = \{a\}$, we can consider that $\mathbf{M} = \{a^i : 0 \leq i \leq p\} \cup \{(a^p)^\sharp\}$ (by aperiodicity of $\mathbf{M}$) and $(a^p)^\sharp \leq a^p$ is the only pair in $\leq$. We can show that for all $b \in \mathbf{M}$, f_b is LTL$^{\leq}$-computable:

- If $i < p$, $f_{a^i} \approx [\![\bigwedge_{0 \leq j < i} X^j a \wedge X^i \Omega]\!]$,
- $f_{a^p} \approx [\![\bot U^{\leq N} \Omega]\!]$,
- $f_{(a^p)^\sharp} \approx [\![\bigwedge_{0 \leq j < p} X^j a]\!]$

Let us assume that $|\mathbf{M}| > 1$, $|\mathbb{A}| > 1$, and the theorem is true for all $(|\mathbf{M}'|, |\mathbb{A}'|) <_{lex} (|\mathbf{M}|, |\mathbb{A}|)$. We choose a letter $b \neq 1 \in \mathbb{A}$, let $\mathbb{B} = \mathbb{A} \setminus \{b\}$.

Let $L_0 = \mathbb{B}^*$, $L_1 = \mathbb{B}^* b \mathbb{B}^*$, and $L_2 = \mathbb{B}^* b (\mathbb{B}^* b)^+ \mathbb{B}^*$. We have $\mathbb{A}^* = L_0 \cup L_1 \cup L_2$.

We define restrictions of f_m: f_0, f_1, f_2 on L_0, L_1, L_2 respectively (giving value ∞ outside of the domain). We have $f_m = \min(f_0, f_1, f_2)$. Hence it suffices to show that the f_i's are LTL$^{\leq}$-computable to get that f_m is also LTL$^{\leq}$-computable (always up to $\approx$).

f_0 is computed by $\mathbf{M}$ on alphabet $\mathbb{B}$, so by induction hypothesis there is a formula φ_0 on $\mathbb{B}$ computing f_0. The formula $\varphi_0' = \varphi_0 \wedge G \neg b$ is a formula on $\mathbb{A}$ computing f_0.

For all $x \in \mathbf{M}$, let φ_x be the LTL$^{\leq}$-formula on $\mathbb{B}$ computing f_x (restricted to $\mathbb{B}^*$), these formulae exist by induction hypothesis, since $|\mathbb{B}| < |\mathbb{A}|$.

If φ is an LTL$^{\leq}$-formula on $\mathbb{B}$, we define its "relativisation" φ' on $\mathbb{A}$ which has the effect of φ on the part before b in a word. We define φ' by induction in the following way:

$$\begin{array}{lcl} a' & = & a \wedge XFb \\ \Omega' & = & b \\ (\varphi \wedge \psi)' & = & \varphi' \wedge \psi' \\ (\varphi \vee \psi)' & = & \varphi' \vee \psi' \\ (X\varphi)' & = & X\varphi' \wedge \neg b \\ (\varphi U \psi)' & = & (\varphi' \wedge \neg b) U \psi' \\ (\varphi U^{\leq N} \psi)' & = & (\varphi' \wedge \neg b) U^{\leq N} \psi' \end{array}$$

With this definition, $[\![\varphi']\!](u_1 b u_2) = [\![\varphi]\!](u_1)$ for any $u_1 \in \mathbb{B}^*$ and $u_2 \in \mathbb{A}^*$.

We define the following formula on $\mathbb{A}$:

$$\varphi_1 = \Big(\bigvee_{xby=m} (\varphi'_x \wedge F(b \wedge X\varphi_y)) \wedge (\neg b U (b \wedge XG\neg b))$$

The second part controls that the word is in L_1. We show $[\![\varphi_1]\!] \approx f_1$.

Let $u \in L_1$, we can write $u = u_1 b u_2$ with $u_1, u_2 \in \mathbb{B}^*$.

By definition of φ_1,

$$\begin{array}{ll} [\![\varphi_1]\!](u) & = \min_{xby=m} \max([\![\varphi'_x]\!](u), [\![\varphi_y]\!](u_2)) \\ & = \min_{xby=m} \max([\![\varphi_x]\!](u_1), [\![\varphi_y]\!](u_2)) \\ & = \min_{xby=m} \max(f_x(u_1), f_y(u_2)). \end{array}$$

We have for any $z \in \mathbf{M}$ and $v \in \mathbb{B}^*$, $\rho(v) \succeq \perp|_{f_z(v)} z$ where $\perp$ is an extra smallest element (by definition of f_z).

But for any x, y such that $xby = m$,

$$\begin{array}{ll} \rho(u) & \sim \tilde{\rho}(\rho(u_1) b \rho(u_2)) \\ & \succeq \tilde{\rho}(\perp|_{f_x(u_1)} x \cdot b \cdot \perp|_{f_y(u_2)} y) \\ & \succeq \perp|_{\max(f_x(u_1), f_y(u_2)} m. \end{array}$$

It implies that for some β (not depending on u), $\forall x, y$ such that $xby = m$, $f_m(u) \leq_\beta \max(f_x(u_1), f_y(u_2))$.

In particular, $f_1(u) = f_m(u) \leq_\beta \min_{xby \in I} \max(f_x(u_1), f_y(u_2)) = [\![\varphi_1]\!](u)$. We can conclude $f_1 \preccurlyeq [\![\varphi_1]\!]$.

Conversely, let us assume that $f_1(u) \leq n$, it means that $\rho(u)(n) \geq m$. but $\rho(u) \sim_\alpha \rho(u_1) \cdot b \cdot \rho(u_2)$, so $\rho(u_1)(\alpha(n)) \cdot b \cdot \rho(u_2)(\alpha(n)) \geq m$.

Let $x = \rho(u_1)(\alpha(n))$ and $y = \rho(u_2)(\alpha(n))$, we have $f_x(u_1) \leq \alpha(n)$ and $f_y(u_2) \leq \alpha(n)$, so $\max(f_x(u_1), f_y(u_2)) \leq \alpha(n)$. We get $[\![\varphi_1]\!](u) \leq \alpha(n)$, and in conclusion $[\![\varphi_1]\!] \preccurlyeq f_1$. This concludes the proof of $[\![\varphi_1]\!] \approx f_1$.

Last but not least, we have to show that f_2 is LTL$^{\leq}$-computable up to $\approx$. For that we will finally use the induction hypothesis on the size of the monoid (until now we only have decreased the size of the alphabet and kept the monoid unchanged).

We define the stabilization monoid $\mathbf{M'} = \langle Mb \cap bM, \circ, \natural, \leq' \rangle$ in the following way: $xb \circ by = xby$, and for xb idempotent $(xb)^\natural = (x^\omega)^\sharp b$ where $x^\omega = x^{|\mathbf{M}|}$ is idempotent, since $\mathbf{M}$ is aperiodic. $\mathbf{M'}$ is a stabilization monoid, let ρ' be compatible with $\mathbf{M'}$. We can first notice that this definition implies that for all $k \in \mathbb{N}$, $(xb)^k = x^k b$, so $\mathbf{M'}$ is also aperiodic. Moreover, we show that $1 \notin \mathbf{M}$: Assume $1 \in \mathbf{M'}$, let $k = |\mathbf{M}|$, $1 = xb = (xb)^k = x^k b^k = x^k b^{k+1} =$

$(xb)^k b = 1b = b$, but $b \neq 1$ so $1 \notin \mathbf{M}'$. Remark that b is the neutral element for $\circ$ in $\mathbf{M}'$, and $|\mathbf{M}'| < |\mathbf{M}|$, which allows us to use induction hypothesis on $\mathbf{M}'$ with alphabet $\mathbf{M}'$.

Let $\Delta = b(\mathbb{B}^* b)^+$, then $L_2 = \mathbb{B}^* \Delta \mathbb{B}^*$.

Let $d \in \mathbf{M}$, we first want to show that f_d over language Δ is $\mathrm{LTL}^{\leq}$-computable up to $\approx$.

Let $\sigma : \Delta \to (\mathbf{M}'^{\mathbb{N}})^*$ defined by

$$\sigma(bu_1 b \dots u_k b) = (b\rho(u_1)b) \dots (b\rho(u_k)b).$$

By induction hypothesis, for any $x \in \mathbf{M}'$, there exists an $\mathrm{LTL}^{\leq}$-formula ψ_x on alphabet $\mathbf{M}'$ and a correction function α such that for any $v \in \mathbf{M}'^*$,
$[\![\psi_x]\!](v) \approx_\alpha \inf\{n \in \mathbb{N} : \rho'(v)(n) \geq x\}$.

Definition 8.4. Let $\mathbf{S}$ be a stabilization monoid. Let f be a function $\mathbf{S}^* \to \mathbb{N}^\infty$, and $\mathbf{S}^\uparrow$ be the set of α-increasing sequences of elements of $\mathbf{S}$ (for some α). We define $\tilde{f} : \mathbf{S}^\uparrow \to \mathbb{N}_\infty$ by $\tilde{f}(\mathbf{u}) = \inf\{n : f(u_n) \leq n\}$.

Remark that this notation is coherent with the $\tilde{}$ operator previously defined for functions $S^+ \to \mathbb{N} \to S$ in the sense that if $f^{\approx}$ is recognized by $\mathbf{S}, h, I$ with compatible function ρ, i.e. $f \approx u \mapsto I[\rho(h(u))]$, then $\tilde{f} \approx \mathbf{u} \mapsto I[\tilde{\rho}(h(\mathbf{u}))]$.

This definition is needed because we already make use of ρ to define σ, so each word of $\mathbb{B}^*$ is mapped to a sequence of elements. However we will need to recombine these various elements, so we will need a formula which is able to take as input sequences instead of words. This will be obtained by applying the tilde operator to the semantic of a formula.

Lemma 8.5. *We claim that there exists α and ϕ_d an $\mathrm{LTL}^{\leq}$-formula on alphabet $\mathbb{A}$ such that for all $u \in \Delta$ and $v \in \mathbb{B}^*$:*

$$[\![\phi_d]\!](uv) \approx_\alpha \widetilde{[\![\psi_d]\!]}(\sigma(u)) \approx_\alpha f_d(u)$$

Intuitively, ϕ_d forgets the last $\mathbb{B}^*$-component v of its input, and is able to apply $\sigma(u)$ to split the word according to the b's, and compute the value of each component with respect to ψ_d.

With this result we can build a formula φ_2 computing f_2:

$$\varphi_2 = \Big(\bigvee_{xdy=m} (\varphi'_x \wedge F(b \wedge X\phi_d)) \wedge F(b \wedge X(G\neg b \wedge \varphi_y))\Big) \wedge \varphi_{L_2}$$

where $\varphi_{L_2} = F(b \wedge XFb)$ controls that the word is in L_2.

By construction, lemmas and induction hypothesis, there exists α such that for all $v_1, v_2 \in \mathbb{B}^*$ and $u \in \Delta$,

$$\begin{aligned}[\![\varphi_2]\!](v_1 u v_2) &\approx_\alpha \min_{xdy=m} \max([\![\varphi'_x]\!](v_1 u v_2), [\![\phi_d]\!](uv_2), [\![\varphi_y]\!](v_2)) \\ &\approx_\alpha \min_{xdy=m} \max(f_x(v_1), f_d(u), f_y(v_2)).\end{aligned}$$

The proof that $min_{xdy=m} \max(f_x(v_1), f_d(u), f_y(v_2)) \approx f_m(v_1 u v_2)$ is similar to the proof of $[\![\varphi_1]\!] \approx f_1$.

All this together gives us $[\![\varphi_2]\!] \approx f_2$, which concludes the proof. □

Proof of Lemma 8.5.

Proof. First let us show that $\widetilde{[\![\psi_d]\!]}(\sigma(u)) \approx_\alpha f_d(u)$ for some α and all $u \in \Delta$. Let $u = bu_1bu_2 \dots u_kb$ with $u_i \in \mathbb{B}^*$. For each $i \in [\![1,k]\!]$ and $t \in \mathbb{N}$, $\rho(u_i)(t) = a_{i,t} \in \mathbf{M}$. For all $t \in \mathbb{N}$, let $v_t = (ba_{1,t}b)\dots(ba_{k,t}b)$, v_t is a word on $\mathbf{M}'$ of length k, and $\sigma(u) = (v_t)_{t\in\mathbb{N}}$. Finally, let $w_t = ba_{1,t}ba_{2,t}\dots ba_{k,t}b$ of length $2k+1$ on $\mathbf{M}$.

On a :

$$\begin{aligned}\widetilde{[\![\psi_d]\!]}(\sigma(u))) &= \inf\{t : [\![\psi_d]\!](v_t) \le t\}\\ &\approx \inf\{t : \inf\{n : \rho'(v_t)(n) \ge d\} \le t\}\end{aligned}$$

We will show that $\rho'(v_t) \sim \rho(w_t)$ for all t. It suffices to verify that ρ' and ρ both verify all axioms of Theorem 3.6 over $(ba_1b)\dots(ba_kb)$ and $ba_1ba_2\dots a_kb$ respectively. Let α and α' be the correction functions given by this theorem for ρ and ρ'.

Letter.: For all $a \in M$, we have $\rho'(bab) \sim_{\alpha'} bab \sim_\alpha \rho(bab)$,

Produit.: For all $a_1, a_2 \in M$, we have $\rho'((ba_1b)(ba_2b)) \sim_{\alpha'} (ba_1b)\circ(ba_2b) = ba_1ba_2b \sim_{\alpha^4} \rho(ba_1ba_2b)$,

Stabilization.: Let bab be an idempotent of $\mathbf{M}'$, and $m \in \mathbb{N}$, Notice that $babab = bab$, so for all $l \ge 1$, $(ba)^lb = bab$, where product is relative to $\mathbf{M}$.

$\rho'((bab)^m) \sim_{\alpha'} (bab)^\natural|_m(bab) = (ba)^{\omega\sharp}b|_m(bab)$. We perform the euclidean division $m = |\mathbf{M}|m' + m''$ avec $m'' < |\mathbf{M}|$,

$$\begin{aligned}\rho((ba)^mb) &\sim \rho(((ba)^{|\mathbf{M}|})^{m'})\cdot(ba)^{m''}b\\ &\sim^{(1)} \rho(((ba)^\omega)^{m'})\cdot(bab)\\ &\sim (ba)^{\omega\sharp}(bab)|_{m'}(ba)^\omega(bab)\\ &\sim^{(2)} (ba)^{\omega\sharp}b|_m(bab).\end{aligned}$$

Equivalence (1) is obtained by aperiodicity of $\mathbf{M}$ ($(ba)^\omega$ is a letter here, not a word of length $|\mathbf{M}|$), and equivalence (2) by using the property $m \approx_{\times(|\mathbf{M}|+1)} m'$.

These three cases show that any n-tree in $\mathbf{M}'$ over v_t can be transformed into an n-tree over w_t, since each type of node is preserved. The substitution property corresponds to branching several n-trees together, so it is necessary to treat it here.

Thus we obtain $\rho'(v_t) \sim \rho(w_t)$ for all t.

Moreover, let $\mathbf{w} = (w_t)_{t\in\mathbb{N}}$, we want to show the following property:

$$(EQ):\ \inf\left\{n' : \tilde\rho(\mathbf{w})(n') \ge d\right\} \approx \inf\{t : \inf\{n : \rho(w_t)(n) \ge d\} \le t\}.$$

Let $N' = \inf\{n' : \tilde\rho(\mathbf{w})(n') \ge d\}$. Notice that $\rho(w_{N'})(N') \ge d$, and consequently $N' \ge \inf\{t : \inf\{n : \rho(w_t)(n) \ge d\} \le t\}$.

Conversely, let $T = \inf\{t : \inf\{n : \rho(w_t)(n) \ge d\} \le t\}$ and N the corresponding value of $\inf\{n : \rho(w_t)(n) \ge d\}$, we have $N \le T$ and $\rho(w_t)$ is α-increasing, so $\rho(w_T)(T) \ge_\alpha \rho(w_T)(N) \ge d$, i.e. $T \ge_\alpha \inf\{n' : \tilde\rho(\mathbf{w})(n') \ge d\}$.

Hence we have the equivalence (EQ).

Finally,

$$\begin{aligned}\widetilde{[\![\psi_d]\!]}(\sigma(u))) &\approx \inf\{t : \inf\{n : \rho(w_t)(n) \ge d\} \le t\}\\ &\approx \inf\{n : \tilde\rho(\mathbf{w})(n) \ge d\} && \text{by } (EQ)\\ &= \inf\{n : \tilde\rho(b\rho(u_1)b\rho(u_2)\dots\rho(u_k)b)(n) \ge d\}\\ &\approx \inf\{n : \rho(bu_1bu_2\dots u_kb)(n) \ge d\} && \text{Substitution axiom}\\ &\approx f_d(u).\end{aligned}$$

which concludes the proof of $\widetilde{[\![\psi_d]\!]}(\sigma(u))) \approx f_d(u)$.

It remains to show that there exists a formula ϕ_d and an α such that for all $u, v \in \Delta \times \mathbb{B}^*$, $[\![\phi_d]\!](uv) \approx_\alpha \widetilde{[\![\psi_d]\!]}(\sigma(u))$.

If ψ is an LTL$^{\leq}$-formula on $\mathbf{M}'$, we define $\psi^\bigstar$ on alphabet $\mathbb{A}$ by induction on ψ:

$$\begin{array}{ll} x^\bigstar & = (b \wedge XFb) \wedge (X\varphi'_x) \\ (\psi_1 \wedge \psi_2)^\bigstar & = \psi_1^\bigstar \wedge \psi_2^\bigstar \\ (\psi_1 \vee \psi_2)^\bigstar & = \psi_1^\bigstar \vee \psi_2^\bigstar \\ (X\psi)^\bigstar & = \neg bU(b \wedge \psi^\bigstar) \\ (\psi_1 U \psi_2)^\bigstar & = (b \implies \psi_1^\bigstar) U (b \wedge \psi_2^\bigstar) \\ (\psi_1 U^{\leq N} \psi_2)^\bigstar & = (b \implies \psi_1^\bigstar) U^{\leq N} (b \wedge \psi_2^\bigstar). \end{array}$$

Where φ'_x is defined as before for any φ_x on alphabet $\mathbb{B}$.

Let us show by induction on ψ that that $[\![\psi^\bigstar]\!](uv) \approx \widetilde{[\![\psi]\!]}(\sigma(u))$ for $u = bu_1bu_2 \dots u_kb \in \Delta$ and $v \in \mathbb{B}^*$:

- If $x \in \mathbf{M}'$,
  $[\![x^\bigstar]\!](uv) = [\![\varphi'_x]\!](u_1bu_2 \dots u_kbv) = [\![\varphi_x]\!](u_1)$, and
  $\widetilde{[\![x]\!]}(\sigma(u)) = \inf\{n : [\![x]\!](\rho(u_1)(n)) \leq n\} \approx \inf\{n : (\rho(u_1)(n)) \geq x\} \approx [\![\varphi_x]\!](u_1)$.
- $\wedge$ case:

$$\begin{aligned} [\![(\psi_1 \wedge \psi_2)^\bigstar]\!](uv) &= \max([\![\psi_1^\bigstar]\!](uv), [\![\psi_2^\bigstar]\!](uv)) \\ &\approx \max(\widetilde{[\![\psi_1]\!]}(\sigma(u)), \widetilde{[\![\psi_2]\!]}(\sigma(u))) \\ &\approx \widetilde{[\![\psi_1 \wedge \psi_2]\!]}(\sigma(u)) \end{aligned}$$

- $\vee$ case:

$$\begin{aligned} [\![(\psi_1 \vee \psi_2)^\bigstar]\!](uv) &= \min([\![\psi_1^\bigstar]\!](uv), [\![\psi_2^\bigstar]\!](uv)) \\ &\approx \min(\widetilde{[\![\psi_1]\!]}(\sigma(u)), \widetilde{[\![\psi_2]\!]}(\sigma(u))) \\ &\approx \widetilde{[\![\psi_1 \vee \psi_2]\!]}(\sigma(u)) \end{aligned}$$

- X case:

$$\begin{aligned} [\![(X\psi)^\bigstar]\!](uv) &= [\![\psi^\bigstar]\!](bu_2b \dots u_kbv) \\ &\approx \widetilde{[\![\psi]\!]}(\sigma(bu_2b \dots u_kb)) \\ &\approx \widetilde{[\![X\psi]\!]}(\sigma(bu_1bu_2b \dots u_kb)) \end{aligned}$$

- U case:

$$\begin{aligned} [\![(\psi_1 U \psi_2)^\bigstar]\!](uv) &= \min_{1 \leq j \leq k} (\max([\![\psi_2^\bigstar]\!](bu_jb \dots u_kbv), \max_{1 \leq i \leq j} [\![\psi_1^\bigstar]\!](bu_ib \dots u_kbv))) \\ &\approx \min_{1 \leq j \leq k} (\max(\widetilde{[\![\psi_2]\!]}(\sigma(bu_jb \dots u_kb)), \max_{1 \leq i \leq j} \widetilde{[\![\psi_1]\!]}(\sigma(bu_ib \dots u_kb)))) \\ &\approx \widetilde{[\![\psi_1 U \psi_2]\!]}(\sigma(u)) \end{aligned}$$

- The $U^{\leq N}$ case is the same as above, allowing at most N mistakes for ψ_1.

We now just have to take $\phi_d = \psi_d^\bigstar$ to complete the proof of Lemma 8.5. □

Corollary 8.6. *The class of LTL$^{\leq}$-definable cost functions is decidable.*

Proof. Theorems 8.2 and 8.3 imply that it is equivalent for a regular cost function to be LTL$^{\leq}$-definable or to have an aperiodic syntactic stabilization semigroup. If F is given by an automaton or a stabilization semigroup, we can compute its syntactic stabilization semigroup $\mathbf{S}_F$ (see [CKL10]) and decide if F is LTL$^{\leq}$-definable by testing aperiodicity of $\mathbf{S}_F$. This can be done simply by iterating at most $|\mathbf{S}_F|$ times all elements of $\mathbf{S}_F$ and see if each element a reaches an element a^k such that $a^{k+1} = a^k$. □

9. Conclusion

We first defined LTL$^{\leq}$ as a quantitative extension of LTL. We started the study of LTL$^{\leq}$ by giving an explicit translation from LTL$^{\leq}$-formulae to B-automata and S-automata, therefore showing that the boundedness (and comparison) problem for LTL$^{\leq}$-formulae is PSPACE-complete. We then showed that the expressive power of LTL$^{\leq}$ in terms of cost functions is the same as aperiodic stabilization semigroups. The proof uses a new syntactic congruence, which has a general interest in the study of regular cost functions. This result implies the decidability of the LTL$^{\leq}$-definable class of cost functions.

As a further work, we can try to put $\omega\sharp$-expressions in a larger framework, by doing an axiomatization of $\omega\sharp$-semigroups. We can also extend this work to infinite words, and define an analog to Büchi automata for cost functions. To continue the analogy with classic languages results, we can define a quantitative extension of FO describing the same class as LTL$^{\leq}$, and search for analog definitions of counter-free B-automata and star-free B-regular expressions. The translation from LTL$^{\leq}$-formulae to B-automata can be further studied in terms of optimality of number of counters of the resulting B-automaton.

Acknowledgments. I am very grateful to my advisor Thomas Colcombet for our helpful discussions, and for the guidelines he gave me on this work, and to Michael Vanden Boom for helping me with language and presentation issues. I also thank the anonymous referees for their useful comments.

References

[AETP01] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron Peled. Parametric temporal logic for "model measuring". *ACM Trans. Comput. Log.*, 2(3):388–407, 2001.

[BC06] Mikołaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *Proceedings of LICS 2006*, pages 285–296. IEEE Computer Society Press, 2006.

[Boj04] Mikołaj Bojańczyk. A bounding quantifier. In *Computer science logic*, volume 3210 of *Lecture Notes in Comput. Sci.*, pages 41–55. Springer, Berlin, 2004.

[CKL10] Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In *Automata, languages and programming. Part II*, volume 6199 of *Lecture Notes in Comput. Sci.*, pages 563–574. Springer, Berlin, 2010.

[Col09] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150, Berlin, 2009. Springer.

[DG10] Stéphane Demri and Paul Gastin. Specification and verification using temporal logics. In *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2010. To appear.

[DJP04] Nachum Dershowitz, D.N. Jayasimha, and Seungjoon Park. Bounded fairness. In Nachum Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 440–442. Springer Berlin / Heidelberg, 2004.

[Has82] Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.

[Has88] 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.

[Has90] Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theoret. Comput. Sci.*, 72(1):27–38, 1990.

[Kir05] Daniel Kirsten. Distance desert automata and the star height problem. *Theor. Inform. Appl.*, 39(3):455–509, 2005.

[KPV09] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. Assoc. Comput. Mach.*, 32(3):733–749, 1985.

[Sch65] M.-P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control 8*, pages 190–194, 1965.

[VW86] Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

[Wil99] Thomas Wilke. Classifying discrete temporal properties. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1999.