Linear temporal logic for regular cost functions

Denis Kuperberg

Linear temporal logic for regular cost functions

Denis Kuperberg

LIAFA/**CNRS/Université Paris 7, Denis Diderot, France**

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 automata. We then algebraically characterize the expressive power of this logic, using a new syntactic congruence for cost functions introduced in this paper.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.627

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 [2]. 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...

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 [10]. Moreover quantitative extensions of LTL have recently been successfully introduced. For instance the model Prompt-LTL introduced in [8] 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 ($\text{LTL}^{\leq}$), 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).

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

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 627–636
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

resolved by Hashigushi [5] using a very intricate proof, and later by Kirsten [7] using an automaton that has counting features.

Finally, also using ideas inspired from [1], the theory of those automata over words has been unified in [2], 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 [8], is an interesting way to extend LTL in order to look at boundedness issues, and already gave interesting decidability and complexity results. Prompt-LTL would correspond in the framework of regular cost functions to a subclass of temporal cost functions introduced in [3]; in particular it is weaker than $\mathrm{LTL}^{\leq}$ introduced here.

Contributions

It is known from [2] that regular cost functions are the ones recognizable by stabilization semigroups (or in an equivalent way, stabilization monoids), and from [3] 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 $\mathrm{LTL}^{\leq}$-formulas 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 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.

This work validates the algebraic approach for studying cost functions, since the analogy extends to syntactic congruence. It also allows a more user-friendly way to describe cost functions, since LTL can be more intuitive than automata or stabilization semigroups to describe a given cost function.

As it was done in [3] for temporal cost functions, the characterization result obtained here for $\mathrm{LTL}^{\leq}$-definable cost functions follows the spirit of Schützenberger's theorem which links star-free languages with aperiodic monoids [9].

Organisation of the paper

After some notations, and reminder on cost functions, we introduce in Section 3 $\mathrm{LTL}^{\leq}$ as a quantitative extension of LTL, and give an explicit translation from $\mathrm{LTL}^{\leq}$-formulae to B-automata. We then present in Section 4 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 $\mathrm{LTL}^{\leq}$ 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 $0 < 1 < \cdots < \infty$. 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 words). Such sequences will be denoted by bold letters ($\vec{a}$, $\vec{b}$,...). 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 $|u|_a$. Functions $\mathbb{N} \to \mathbb{N}$ will be denoted by letters $\alpha, \beta, \ldots$, and will be extended to $\mathbb{N} \cup \{\infty\}$ by $\alpha(\infty) = \infty$.

2 Regular Cost functions

2.1 Cost functions and equivalence

If $L \subseteq \mathbb{A}^*$, we will note χ_L the function defined by $\chi_L(u) = 0$ if $u \in L$, ∞ if $u \notin L$. Let $\mathcal{F}$ be the set of functions : $\mathbb{A}^* \to \mathbb{N}_\infty$. For $f, g \in \mathcal{F}$ and α a 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 $g \leq_\alpha f$. Finally $f \approx g$ if $f \approx_\alpha g$ for some α. This equivalence relation doesn't pay attention to exact values, but preserves the existence of bounds.

A *cost function* is an equivalence class of $\mathcal{F}/\approx$. Cost functions are noted $f, g, \ldots$, and in practice they will be always be represented by one of their elements in $\mathcal{F}$.

2.2 B-automata

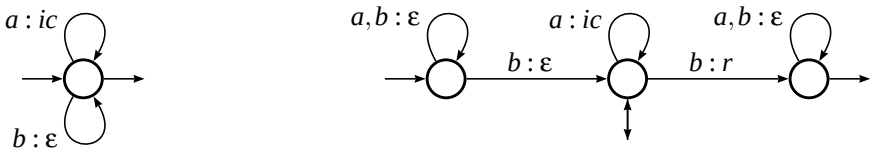
A B-automaton is a tuple $\langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta \rangle$ where Q is the set of states, $\mathbb{A}$ the alphabet, In and Fin the sets of initial and final states, Γ the set of counters, and $\Delta \subseteq Q \times \mathbb{A} \times (\{i, r, c\}^*)^\Gamma \times Q$ is the set of transitions.

Counters have integers values starting at 0, and an action $\sigma \in (\{i, r, c\}^*)^\Gamma$ performs a sequence of atomic actions on each counter, where atomic actions are either i (increment by 1), r (reset to 0) or c (check the value). In particular we will note ε the action corresponding to the empty word : doing nothing on every counter. If e is a run, let $C(e)$ be the set of values checked during e on all counters of Γ.

A B-automaton $\mathcal{A}$ computes a regular cost function $[\![\mathcal{A}]\!]$ via the following semantic : $[\![\mathcal{A}]\!](u) = \inf\{\sup C(e), e \text{ run of } \mathcal{A} \text{ over } u\}$.

With the usual conventions that $\sup \emptyset = 0$ and $\inf \emptyset = \infty$. There exists also a dual model of B-automata, namely S-automata, that has the same expressive power, but we won't develop this further in this paper. See [2] for more details.

▶ Example 1. Let $\mathbb{A} = \{a, b\}$. The cost function $|\cdot|_a$ is the same as $2|\cdot|_a + 5$, it is computed by the following one-counter B-automaton on the left-hand side. The cost function $u \mapsto \min\{n \in \mathbb{N},\ a^n \text{ factor of } u\}$ is computed by the nondeterministic one-counter B-automaton on the right-hand side.



Moreover, as in the case of languages, cost functions can be recognized by an algebraic structure that extends the classic notion of semigroups, called stabilization semigroups. A stabilization semigroup $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ is a partially ordered set S together with an internal binary operation $\cdot$ and an internal unary operation $a \mapsto a^\sharp$ defined only on idempotent elements (elements a such that $a \cdot a = a$). The formalism is quite heavy, see appendix for all details on axioms of stabilization semigroups and recognition of regular cost functions.

3 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?

3.1 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 $U^{\leq N}$. Unlike in most uses of LTL, we work here over finite words.

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

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

Note the absence of negation in the definition of LTL$^{\leq}$. The negations have been pushed to the leaves.

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

We can define $\top = (\bigvee_{a \in \mathbb{A}} a) \vee \Omega$ and $\bot = \neg\top$, meaning respectively true and false, and $\neg a = (\bigvee_{b \neq a} b) \vee \Omega$ to signify that the current letter is not a.

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

3.2 Semantics

We want to associate a cost function $[\![\phi]\!]$ on words to any LTL$^{\leq}$-formula ϕ.

We will say that $u, n \models \phi$ (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

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

We can remark that if $u, n \models \phi$, then for all $k \geq n, u, k \models \phi$, since the $U^{\leq N}$ operators appear always positively in the formula (that is why we don't allow the negation of an LTL$^{\leq}$-formula in general). In particular, $[\![\phi]\!](u) = 0$ means that $\forall n \in \mathbb{N}, u, n \models \phi$, and $[\![\phi]\!](u) = \infty$ means that $\forall n \in \mathbb{N}, u, n \not\models \phi$ (since $\inf \emptyset = \infty$).

▶ Proposition 2.

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

▶ Example 3. Let $\phi = (\neg a) U^{\leq N} \Omega$, then $[\![\phi]\!] = |\cdot|_a$

We use LTL$^{\leq}$-formulae in order to describe cost functions, so we will always work modulo cost function equivalence $\approx$.

▶ Remark 4. If ϕ does not contain any operator $U^{\leq N}$, ϕ is a classic LTL-formula computing a language L, and $[\![\phi]\!] = \chi_L$.

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

We will now give a direct translation from LTL$^{\leq}$-formula 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 $[\![\phi]\!]$. This construction is adapted from the classic translation from LTL-formula to Büchi automata [4].

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

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 $\Gamma = \{\gamma_1, \dots, \gamma_k\}$ 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. 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 6 (Next Step). If Z is consistent and reduced, for all $u \in \mathbb{A}^*, a \in \mathbb{A}$ and $n \in \mathbb{N}$,

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

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 don't (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.

▶ Definition 7. 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 8. Let $u = a_1 \dots a_m$ be a word on $\mathbb{A}$ and $Y_0 \xrightarrow{a_1:\sigma_1} Y_1 \xrightarrow{a_2:\sigma_2} \dots \xrightarrow{a_m:\sigma_m} Y_m$ an accepting run of $\mathcal{A}_\phi$.

Then for all $\psi \in \mathrm{sub}(\phi)$, for all $n \in \{0,\dots,m\}$, for all $Y_n \xrightarrow{\varepsilon:\sigma}_* Y \xrightarrow{\varepsilon:\sigma'}_* Z$ with $Z \cup \{a_{n+1}\}$ 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 $N = \mathrm{val}(\sigma'\sigma_{n+1}\dots\sigma_m)$.

Lemma 8 implies the correctness of the automaton $\mathcal{A}_\phi$:
Let $Y_0 \xrightarrow{a_1:\sigma_1} Y_1 \xrightarrow{a_2:\sigma_2} \dots \xrightarrow{a_m:\sigma_m} Y_m$ be a valid run of $\mathcal{A}_\phi$ on u of value $N = [\![\mathcal{A}_\phi]\!]_B$, applying Lemma 8 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 $[\![\mathcal{A}_\phi]\!]_B \approx [\![\phi]\!]$).

4 Algebraic characterization

We remind 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 [3].

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.

4.1 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 won't define our equivalence over words, but over $\sharp$-expressions, which are a way to describe words with quantitative information.

4.2 $\sharp$-expressions

We first define general $\sharp$-expressions as in [6] and [3] 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 eval function (from Expr to $\mathbf{S}$) is defined inductively by $\text{eval}(a) = h(a), \text{eval}(ee') = \text{eval}(e) \cdot \text{eval}(e')$, and $\text{eval}(e^\sharp) = \text{eval}(e)^\sharp$ ($\text{eval}(e)$ has to be idempotent). We say that e is *well-formed for* $\mathbf{S}$ if $\text{eval}(e)$ exists. Intuitively, it means that $\sharp$ was applied to subexpressions that corresponds to idempotent elements in $\mathbf{S}$.

If f is a regular cost function, e is *well-formed for* f iff e is well-formed for the minimal stabilization semigroup of f.

▶ Example 9. Let f be the cost 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 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 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.

4.3 $\omega\sharp$-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 be well-typed for the cost function f we want to study, and therefore does not correspond to an element in the syntactic stabilization semigroup of f.

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

▶ Lemma 11. Let f be a regular cost function, there exists $K_f \in \mathbb{N}$ such that for any $E \in \text{Oexpr}$, 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. The proof is a little technical, since we have to reuse the definition of recognization by stabilization semigroup. K_f can simply be taken to be the size of the minimal stabilization semigroup of f. ◀

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 12. 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

$$E \equiv_f E' \text{ iff } \forall C[x] \in \text{C}_{\text{OE}}, C[E] \rightleftharpoons_f C[E']$$

▶ Remark 13. 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. $\text{Oexpr}/\equiv_f$ is the syntactic stabilization semigroup of f.

Indeed if f is a regular cost function, let $\mathbf{S}_f = \text{Oexpr}/\equiv_f$. We can provide $\mathbf{S}_f$ with a structure of stabilization semigroup $\langle \mathbf{S}_f, \cdot, \leq, \sharp \rangle$.

▶ **Theorem 14.** *$\mathbf{S}_f$ is the minimal stabilization semigroup recognizing f.*

The proof consists basically in a bijection between classes of Oexpr for $\equiv_f$, and elements of the minimal stabilization semigroup as defined in appendix A.7 of [3].

4.4 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 15. 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.

▶ **Theorem 16.** *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.

If ϕ is a LTL$^\leq$-formula, we will say that ϕ verifies property *AP* if there exists $k \in \mathbb{N}$ such that for any $\omega\sharp$-expression E, $E^k \equiv_{[\![\phi]\!]} E^{k+1}$, which is equivalent to "$[\![\phi]\!]$ has an aperiodic syntactic stabilization semigroup".

With this in mind, we can do an induction on LTL$^\leq$-formulaes : we first show that $\mathbf{S}_\Omega$ and all $\mathbf{S}_a$ for $a \in \mathbb{A}$ are aperiodic.

We then proceed to the induction on ϕ : assuming that φ and ψ verify property *AP*, we show that $X\psi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi U \psi$ and $\varphi U^{\leq N} \psi$ verify property *AP*.

▶ **Theorem 17.** *Let f be a cost function recognized by an aperiodic stabilization semigroup, then f can be described by an LTL$^{\leq}$-formula.*

The proof of this theorem is a generalization of the proof of Wilke for aperiodic languages in [11]. However difficulties inherent to quantitative notions appear here.

The main issue comes from the fact that in the classical setting, computing the value of a word in a monoid returns a single element. This fact is used to do an induction on the size of the monoid, by considering the set of possible results as a smaller monoid. The problem is that with cost functions, there is some additional quantitative information, and we need to associate a sequence of elements of a stabilization monoid to a single word. Therefore, it requires some technical work to come back to a smaller stabilization monoid from these sequences.

▶ Corollary 18. The class of LTL$^{\leq}$-definable cost functions is decidable.

Proof. Theorems 16 and 17 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 [3]) 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 $a^{k+1} = a^k$. ◀

5 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, which preserves exact values (and not only boundedness properties as it is usually the case in the framework of cost functions). 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.

References

1 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, August 2006.

2 Thomas Colcombet. The theory of stabilization monoids and regular cost functions. *ICALP, Lecture Notes in Computer Science*, 2009.

3 Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In *ICALP (2)*, pages 563–574, 2010.

4 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.

5 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.

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

7 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.

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

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

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

11 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.