REGULAR COST FUNCTIONS, PART I: LOGIC AND ALGEBRA OVER WORDS

THOMAS COLCOMBET

Laboratoire Liafa, Universit Paris Diderot - Paris 7, Case 7014, F-75205 Paris Cedex 13, France
e-mail address: thomas.colcombet@liafa.jussieu.fr

Abstract. The theory of regular cost functions is a quantitative extension to the classical notion of regularity. A cost function associates to each input a non-negative integer value (or infinity), as opposed to languages which only associate to each input the two values 'inside' and 'outside'. This theory is a continuation of the work on distance automata and similar models. These models of automata have been successfully used for solving the star-height problem, the finite power property, the finite substitution problem, the relative inclusion star-height problem and the boundedness problem for monadic-second order logic over words. Our notion of regularity can be – as in the classical theory of regular languages – equivalently defined in terms of automata, expressions, algebraic recognisability, and by a variant of the monadic second-order logic. Those equivalences are strict extensions of the corresponding classical results.

The present paper introduces the cost monadic logic, the quantitative extension to the notion of monadic second-order logic we use, and show that some problems of existence of bounds are decidable for this logic. This is achieved by introducing the corresponding algebraic formalism: stabilisation monoids.

The paper is under submission. Version of November 2011.

1. Introduction

This paper introduces and studies a quantitative extension to the standard theory of regular languages of words. It is the only quantitative extension (in which quantitative means that the function described can take infinitely many values) known to the author in which the milestone equivalence for regular languages:

$$\text{accepted by automata} = \text{recognisable by monoids}$$
$$= \text{definable in monadic second-order logic} = \text{definable by regular expressions}$$

can be faithfully extended.

This theory is developed in several papers. The objective of the present one is the introduction of the logical formalism, and its resolution using algebraic tools. However, in this introduction, we try to give a broader panorama of the whole theory.

Supported by the Anr project Jade: 'Jeux et Automates, Décidabilité et Extensions'. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n 259454.

1.1. **Related works.** The theory of regular cost functions involves the use of automata (called B- and S-automata), algebraic structures (called stabilisation monoids), a logic (called cost monadic logic), and suitable regular expressions (called B- and S-regular expressions). All those models happen to be equi-expressive. Though most of these concepts are new, some are very close to objects known from the literature. As such, the present work is the continuation of long branch of research.

The general idea behind these works is that we want to represent functions, i.e., quantitative variants of languages, and that, ideally we want to keep strong decision results. Works related to cost functions go in this direction, where the quantitative notion is the ability to count, and the decidability results are concerned with the existence/non-existence of bounds.

A prominent question in this theory is the star-height problem. This story begins in 1963 when Eggan formulates the star-height decision problem [13]:

Input: A regular language of words L and a non-negative integer k.

Output: Yes, if there exists a regular expression[1] using at most k nesting of Kleene stars which defines L. No, otherwise.

Eggan proved that the hierarchy induced by k does not collapse, but the decision problem itself was quickly considered as central in language theory, and as the most difficult problem in the area.

Though some partial results were obtained by McNaughton, Dejean and Schützenberger [36, 12], it took twenty-five years before Hashiguchi came up with a proof of decidability spread over four papers [18, 17, 19, 20]. This proof is notoriously difficult, and no clean exposition of it has ever been presented.

Hashiguchi used in his proof the model of *distance automata*. A distance automaton is a finite state non-deterministic automaton running over words which can count the number of occurrences of some 'special' states. Such an automaton associates to each word a natural number, which is the least number of occurrences of special states among all the accepting runs (or nothing if there is no accepting run over this input). The proof of Hashiguchi relies on a very difficult reduction to the following *limitedness* problem:

Input: A distance automaton.

Output: Yes, if the automaton is *limited*, i.e., if the function it computes is bounded over its domain. No, otherwise.

Hashiguchi established the decidability of this problem [17]. The notion of distance automata and its relationship with the tropical semiring (distance automata can be seen as automata over the tropical semiring, i.e, the semiring $(\mathbb{N}\cup\{\infty\}, \min, +)$) has been the source of many investigations [18, 21, 23, 34, 35, 40, 41, 43, 46, 47].

Despite this research, the star-height problem itself remained not so well understood for seventeen more years. In 2005, Kirsten gave a much simpler and self-contained proof [26]. The principle is to use a reduction to the limitedness problem for a form of automata more general than distance automata, called *nested distance desert automata*. To understand this extension, let us first look again at distance automata: we can see a distance automaton as an automaton that has a counter which is incremented each time a 'special' state is encountered. The value attached to a word by such an automaton is the minimum over

[1]Regular expressions are built on top of letters using the language operation of concatenation, union, and Kleene star. This problem is sometime referred to as the restricted star-height problem, while the version also allowing complement is the generalised star-height problem, and has a very different status.

all accepting runs of the maximal value assumed by the counter. Presented like this, a nested distance desert automaton is nothing but a distance automaton in which multiple counters and reset of the counters are allowed (with a certain constraint of nesting of counters). Kirsten performed a reduction of the star-height problem to the limitedness of nested distance desert automata which is much easier than the reduction of Hashiguchi. He also proves that the limitedness problem of nested distance desert automata is decidable. For this, he generalises the proof methods developed previously by Hashiguchi, Simon and Leung for distance automata. This work closes the story of the star-height problem itself.

The star-height problem is the king among the problems solved using this method. But there are many other (difficult) questions that can be reduced to the limitedness of distance automata and variants. Some of the solutions to those problems paved the way to the solution of the star-height problem.

The *finite power property* takes as input a regular language L and asks whether there exists some positive integer n such that $(L+\varepsilon)^n = L^*$. It was raised by Brzozowski in 1966, and it took twelve years before being independently solved by Simon and Hashiguchi [40, 16]. This problem is easily reduced to the limitedness problem for distance automata.

The *finite substitution problem* takes as input two regular languages L, K, and asks whether it is possible to find a finite substitution σ (i.e., a morphism mapping each letter of the alphabet of L to a finite language over the alphabet of K) such that $\sigma(L) = K$. This problem was shown decidable independently by Bala and Kirsten by a reduction to the limitedness of desert automata (a form of automata weaker than nested distance desert automata, but incomparable to distance automata), and a proof of decidability of this latter problem [2, 25].

The *relative inclusion star-height problem* is an extension of the star height problem introduced and shown decidable by Hashiguchi using his techniques [22]. Still using nested distance desert automata, Kirsten gave another, more elegant proof of this result [29].

The *boundedness problem* is a problem of model theory. It consists of deciding if there exists a bound on the number of iterations that are necessary for the fixpoint of a logical formula to be reached. The existence of a bound means that the fixpoint can be eliminated by unfolding its definition sufficiently many times. The boundedness problem is usually parameterised by the logic chosen and by the class of models over which the formula is studied. The boundedness problem for monadic second-order formulae over the class of finite words was solved by a reduction to the limitedness problem of distance automata by Blumensath, Otto and Weyer [3].

One can also cite applications of distance automata in speech recognition [37, 38], databases [14], and image compression [24]. In the context of verification, Abdulla, Krcàl and Yi have introduced R-automata, which correspond to nested distance desert automata in which the nesting of counters is not required anymore [1]. They prove the decidability of the limitedness problem for this model of automata.

Finally, Löding and the author have also pursued this branch of researches in the direction of extended models. In [9], the *star-height problem over trees* has been solved, by a reduction to the limitedness problem of nested distance desert automata over trees. The latter problem was shown decidable in the more general case of alternating automata. In [10] a similar attempt has been tried for deciding the Mostowski hierarchy of non-deterministic automata over infinite trees (the hierarchy induced by the alternation of fixpoints). The authors show that it is possible to reduce this problem to the limitedness problem for a form of automata that unifies nested distance desert automata and parity tree automata. The latter problem is an important open question.

Bojańczyk and the author have introduced the notion of B-automata in [5], a model which resembles much (and is prior to) R-automata. The context was to show the decidability of some fragments of the logic MSO+U over infinite words, in which MSO+U is the extension of the monadic second order logic extended with the quantifier $\mathbb{U}X.\phi$ meaning "for all integer n, there exists a set X of cardinality at least n such that ϕ holds". From the decidability results in this work, it is possible to derive every other limitedness results. However, the constructions are complicated and of non-elementary complexity. Nevertheless, the new notion of S-automata was introduced, a model dual to B-automata. Recall that the semantics of distance automata and their variants can be expressed as a minimum over all runs of the maximum of the value taken by counters. The semantics of S-automata is dual: it is defined as the maximum over all runs of the minimum of the value taken by the counters at the moment of their reset. Unfortunately, it is quite hard to compare in detail this work with all others. Indeed, since it was oriented toward the study of a logic over infinite words, the central automata are in fact ωB and ωS-automata: automata accepting languages of infinite words that have an infinitary accepting condition constraining the asymptotic behaviour of the counters along the run. This makes these automata very different. Indeed, the automata in [5] accept languages while the automata in study here define functions. For achieving this, the automata uses an extra mechanism involving the asymptotic behaviors of counters for deciding whether an infinite word should be accepted or not. This extra mechanism has no equivalent in distance automata, and is in some sense 'orthogonal' to the machinery involved in cost functions. For this reason, B-automata and S-automata in [5] are just intermediate objects that do not have all the properties we would like. In particular B- and S-automata in [5] are not equivalent. However, the principle of using two dual forms of automata is an important concept in the theory of regular cost functions. The study of MSO+U has been pursued in several directions. Indeed, the general problem of the satisfaction of MSO+U is a challenging open problem. One partial result concerns the decision of WMSO+U (the weak fragment in which only quantifiers over finite sets are allowed) which is decidable [4]. However, the techniques involved in this work are not directly related to cost functions.

The proof methods for showing the decidability of the limitedness problem of distance automata and their variants, are also of much interest by themselves. While the original proof of Hashiguchi is quite complex, a major advance has been achieved by Leung who introduced the notion of stabilisation [32, 33] (see also [41] for an early overview). The principle is to abstract the behaviour of the distance automaton in a monoid, and further describe the semantics of the counter using an operator of stabilisation, i.e., an operator which describes, given an element of the monoid, what would be the effect of iterating it a "lot of times". This key idea was further used and refined by Simon, Leung, Kirsten, Abdulla, Krcàl and Yi. This idea was not present in [5], and this is one explanation for the bad complexity of the constructions.

Another theory related to cost functions is the one developed by Szymon Toruńczyk in his thesis [45]. The author proposes a notion of recognisable languages of *profinite words* which happen to be equivalent to cost functions. Indeed, profinite words are infinite sequences of finite words (which are convergent in a precise topology, the profinite topology). As such, a single profinite word can be used as a witness that a function is not bounded. Following the principle of this correspondence, one can see a cost function as a set of profinite words: the profinite words corresponding to infinite sequences of words over which the function is bounded. This correspondence makes Toruńcyk's approach equi-expressive with cost functions over finite words as far as decision questions are concerned. Seen like this, this approach can be seen as the theory of cost functions presented in a more abstract setting. Still, some differences have to be underlined. On one side, the profinite approach, being more abstract, loses some precision. For instance in the present work, we have a good understanding of the precision of the constructions: namely each operation can be performed doing an at most "polynomial approximation[2]". On the other side, the presentation in terms of profinite languages eliminates the corresponding annoying details in the development of cost functions: namely there is no more need to control the approximation at each step. Another interesting point is that the profinite presentation points naturally to extensions, which are orthogonal to cost functions, and are highly related to MSO+U. For the moment, the profinite approach has been developed for finite words only. It is not clear for now how easy this abstract presentation can be used for treating more complex models, as it has been done for cost functions, e.g. over finite trees [11].

1.2. **Survey of the theory.**

The theory of regular cost functions gives a unified and general framework for explaining all objects, results and constructions presented above (apart from the results in [5] that are of a slightly different nature). It also allows to derive new results.

Let us describe the contributions in more details.

Cost functions. The standard notion of language is replaced by the new notion of cost function. For this, we consider mappings from a set E to $\mathbb{N} \cup \{\infty\}$ (in practice E is the set of finite words over some finite alphabet) and the equivalence relation $\approx$ defined by $f \approx g$ if:

for all $X \subseteq E$, f restricted to X is bounded iff g restricted to X is bounded.

Hence two functions are equivalent if it is not possible to distinguish them using arguments of existence of bounds. A *cost function* is an equivalence class for $\approx$. The notion of cost functions is what we use as a quantitative extension to languages. Indeed, every language L can be identified with (the equivalence class of) the function mapping words in L to the value 0, and words outside L to ∞. All the theory is presented in terms of cost functions. This means that all equivalences are considered modulo the relation $\approx$.

Cost automata. A first way to define regular cost functions is to use cost automata, which come in two flavours, B- and S-automata. The B-automata correspond in their simple form to R-automata [1] and in their simple and hierarchical form to nested distance desert automata in [27, 28]. Those are also very close to B-automata in [5]. Following the ideas in [5], we also use the dual variant of S-automata. The two forms of automata, B-automata and S-automata, are equi-expressive in all their variants, an equivalence that we call the *duality theorem*. Automata are not introduced in this paper.

Stabilisation monoids. The corresponding algebraic characterisation makes use of the new notion of stabilisation monoids. A stabilisation monoid is a finite ordered monoid together with a stabilisation operation. This stabilisation operation expresses what it means to iterate 'a lot of times' some element. The operator of stabilisation was introduced by Leung

[2]The notion of approximation may be misleading: the results are exact, but, since we are only interested in boundedness questions, we allow ourselves to perform some harmless distortions of the functions. This distortion is measured by an approximation parameter called the correction function.

[32, 33] and used also by Simon, Kirsten, Abdulla, Krcàl and Yi as a tool for analysing the behaviour of distance automata and their variants. The novelty here lies in the fact that in our case, stabilisation is now part of the definition of a stabilisation monoid. We prove that it is possible to associate unique semantics to all stabilisation monoids. These semantics are represented by means of computations. A computation is an object describing how a word consisting of elements of the stabilisation monoid can be evaluated into a value in the stabilisation monoid. This key result shows that the notion of stabilisation monoid has a 'meaning' independent from the existence of cost automata (in the same way a monoid can be used for recognising a language, independently from the fact that it comes from a finite state automaton). This notion of computations is easier to handle than the notion of compatible mappings used in the conference version of this work [6].

Recognisable cost functions. We use stabilisation monoids for defining the new notion of recognisable cost functions. We show the closure of recognisable cost functions under min, max, and new operations called inf-projection and sup-projection (which are counterparts to projection in the theory of regular languages). We also prove that the relation $\approx$ (in fact the correspoding preorder $\preccurlyeq$) is decidable over recognisable cost functions. This decidability result subsumes many limitedness results from the literature. This notion of recognisability for cost functions is equivalent to being accepted by the cost automata introduced above.

Extension of regular expressions. It is possible to define two forms of expressions, B- and S-regular expressions, and show that these are equivalent to cost automata. These expressions were already introduced in [5] in which a similar result was established.

Cost-monadic logic. The cost-monadic logic is a quantitative extension to monadic second-order logic. It is for instance possible to define the diameter of a graph in cost-monadic logic. The cost functions over words definable in this logic coincide with the regular cost functions presented above. This equivalence is essentially the consequence of the closure properties of regular cost functions (as in the case of regular languages), and no new ideas are required here. The interest lies in the logic itself. Of course, the decision procedure for recognisable cost function entails decidability results for cost-monadic logic. In this paper, cost-monadic logic is the starting point of our presentation, and our central decidability result is Theorem 2.3 stating the decidability of this logic.

1.3. **Content of this paper.**

This paper does not cover the whole theory of regular cost functions over words. The line followed in this paper is to start from the logic 'cost-monadic logic', and to introduce the necessary material for 'solving it over words'. This requires the complete development of the algebraic formalism.

In Section 2, we introduce the new formalism of cost-monadic logic, and show what is required to solve it. In particular, we introduce the notion of cost function, and advocate that it is useful to consider the logic under this view. We state there our main decision result, Theorem 2.3.

In Section 3 we present the underlying algebraic structure: stabilisation monoids. We then introduce computations, and establish the key results of existence (Theorem 3.3) and uniqueness (Theorem 3.4) of the value computed by computations.

In Section 4, we use stabilisation monoids for defining recognisable cost functions. We show various closure results for recognisable cost functions as well as decision procedures.

Those results happen to fulfill the conditions required in Section 2 for showing the decidability of cost-monadic logic over words.

2. Logic

2.1. **Cost monadic logic.** Let us recall that monadic second-order logic (monadic logic for short) is the extension of first-order logic with the ability to quantify over sets (i.e., monadic relations). Formally monadic formulae use *first-order variables* $(x, y, \dots)$, and *monadic variables* $(X, Y, \dots)$, and it is allowed in such formulae to quantify existentially and universally over both first-order and monadic variables, to use every boolean connective, to use the membership predicate $(x \in X)$, and every predicate of the relational structure. We expect from the reader basic knowledge concerning monadic logic.

Example 2.1. The monadic formula $\mathtt{reach}(x, y, X)$ over the signature containing the single binary predicate **edge** (signature of a digraph):

$$\begin{aligned}\mathtt{reach}(x,y,X) ::= \quad & x = y \quad \vee \quad \forall Z.\\ & \big(x \in Z \wedge \forall z, z'.\ (z \in Z \wedge z' \in X \wedge \mathbf{edge}(z,z') \to z' \in Z)\big) \quad \to \quad y \in Z\end{aligned}$$

describes the existence of a path in a digraph from vertex x to vertex y such that all edges appearing in the path end in X. Indeed, it expresses that either the path is empty, or every sets Z containing x and closed under taking edges ending in X, also contains y.

In cost monadic logic, one uses a single extra variable N of a new kind, called the *bound variable*. It ranges over non-negative integers. Cost monadic logic is obtained from monadic logic by allowing the extra predicate $|X| \leq N$ – in which X is some monadic variable and N the bound variable – iff it appears *positively* in the formula (i.e., under the scope of an even number of negations). The semantic of $|X| \leq N$ is, as one may expect, to be satisfied if (the valuation of) X has cardinality at most (the valuation of) N. Given a formula ϕ, we denote by $\mathrm{FV}(\phi)$ its free variables (the bound variable excluded). A formula that has no free-variables (but the bound variable) is called a *sentence*.

We now have to provide a meaning to the formulae of cost monadic logic. We assume some familiarity of the reader with logic terminology. A *signature* consists of a set of symbols $R, S, \dots$. To each symbol is attached a non-negative integer called its *arity*. A (relational) *structure* (over the above signature) $\mathcal{S} = \langle U_{\mathcal{S}}, R^{\mathcal{S}}, \dots, R^{\mathcal{S}} \rangle$ consists of a set $U_{\mathcal{S}}$ called the *universe*, and for each symbol R of arity n of a relation $R^{\mathcal{S}} \subseteq U_{\mathcal{S}}^n$. Given a set of variables F, a valuation of F (over $\mathcal{S}$) is a mapping v which to each monadic variable $X \in F$ associates a set $\mathrm{v}(X) \subseteq U_{\mathcal{S}}$, and to each first-order variable $x \in F$ associates the element $\mathrm{v}(x) \in U_{\mathcal{S}}$. We denote by $\mathrm{v}, X = E$ the valuation v in which X is further mapped to E. Given a cost monadic formula ϕ, a valuation v of its free variable over a structure $\mathcal{S}$ and a non-negative integer n, we express by $\mathcal{S}, \mathrm{v}, n \models \phi$ the fact that the formula ϕ is satisfied over the structure $\mathcal{S}$ with valuation v when the variable N takes the value n. Of course, if ϕ is simply a sentence, we just write $\mathcal{S}, n \models \phi$. We also omit the parameter n when ϕ is a monadic formula.

The positivity required when using the predicate $|X| \leq N$ has straightforward consequences. Namely, for all cost monadic sentences ϕ, all relational structures $\mathcal{S}$, and all valuations v, $\mathcal{S}, \mathrm{v}, n \models \phi$ implies $\mathcal{S}, \mathrm{v}, m \models \phi$ for all $m \geq n$.

Instead of evaluating as true or false as done above, we see a formula of cost monadic logic ϕ of free variables F as associating to each relational structure $\mathcal{S}$ and each valuation v of the free variables a value in $\mathbb{N}\cup\{\infty\}$ defined by:

$$[\![\phi]\!](\mathcal{S},\mathrm{v}) = \inf\{n \ : \ \mathcal{S},\mathrm{v},n \models \phi\}\ .$$

This value can be either a non-negative integer, or ∞ if no valuation of N makes the sentence true. In case of a sentence ϕ, we omit the valuation and simply write $[\![\phi]\!](\mathcal{S})$. Let us stress the link with standard monadic logic in the following fact:

Fact 2.2. For all monadic formula ϕ, and all relational structures $\mathcal{S}$,

$$[\![\phi]\!](\mathcal{S}) = \begin{cases} 0 & \text{if } \mathcal{S}\models\phi \\ \infty & \text{otherwise .}\end{cases}$$

Example 2.3. The sentence $\forall X.|X| \leq N$ calculates the size of a structure. More formally $[\![\forall X.|X| \leq N]\!](\mathcal{S})$ equals $|U_{\mathcal{S}}|$.

A more interesting example makes use of Example 2.1. Again over the signature of digraphs, the cost monadic sentence:

$$\texttt{diameter} ::= \forall x,y.\exists X.|X| \leq N \wedge \texttt{reach}(\texttt{x},\texttt{y},\texttt{X}).$$

defines the diameter of the di-graph: indeed, the diameter of a graph is the least n such that for all pair of states x, y, there exists a set of size at most N allowing to reach y from x (recall that in the definition of $\texttt{reach}(x,y,X)$, x does not necessarily belong to X, hence this is the diameter in the standard sense).

From now on, for avoiding some irrelevant considerations, we will consider the variant of cost monadic logic in which a) only monadic variables are allowed, b) the inclusion relation $X \subseteq Y$ is allowed, and c) each relation over elements is raised to a relation over singleton sets. Keeping in mind that each element can be identified with the unique singleton set containing it, it is easy to translate cost monadic logic into this variant. In this presentation, it is also natural to see the inclusion relation as any other relation. We will also assume that the negations are pushed to the leaves of formulae as is usual. Overall a formula can be of one of the following forms:

$$R(X_1,\dots X_n) \quad | \quad \neg R(X_1,\dots X_n) \quad | \quad |X|\leq N \quad | \quad \phi\wedge\psi \quad | \quad \phi\vee\psi \quad | \quad \exists X.\phi \quad | \quad \forall X.\phi$$

in which ϕ and ψ are formulas, R is some symbol of arity n which can possibly be $\subseteq$ (of arity 2), and $X, X_1, \dots, X_n$ are monadic variables.

So far, we have described the semantic of cost monadic logic from the standard notion of model. There is another equivalent way to describe the meaning of formulae; namely by induction on the structure. The equations are disclosed in the following fact.

Fact 2.4. Over a structure $\mathcal{S}$ and a valuation v, the following equalities hold:

$$\llbracket R(X_1,\dots,X_n)\rrbracket(\mathcal{S},\mathrm{v}) = \begin{cases} 0 & \text{if } R^{\mathcal{S}}(\mathrm{v}(X_1),\dots,\mathrm{v}(X_n)) \\ \infty & \text{otherwise} \end{cases}$$
$$\llbracket \neg R(X_1,\dots,X_n)\rrbracket(\mathcal{S},\mathrm{v}) = \begin{cases} \infty & \text{if } R^{\mathcal{S}}(\mathrm{v}(X_1),\dots,\mathrm{v}(X_n)) \\ 0 & \text{otherwise} \end{cases}$$
$$\llbracket |X|\leq N\rrbracket(\mathcal{S},\mathrm{v}) = |\mathrm{v}(X)|$$
$$\llbracket \phi\vee\psi\rrbracket(\mathcal{S},\mathrm{v}) = \min(\llbracket\phi\rrbracket(\mathcal{S},\mathrm{v}),\llbracket\psi\rrbracket(\mathcal{S},\mathrm{v}))$$
$$\llbracket \phi\wedge\psi\rrbracket(\mathcal{S},\mathrm{v}) = \max(\llbracket\phi\rrbracket(\mathcal{S},\mathrm{v}),\llbracket\psi\rrbracket(\mathcal{S},\mathrm{v}))$$
$$\llbracket \exists X.\phi\rrbracket(\mathcal{S},\mathrm{v}) = \inf\{\llbracket\phi\rrbracket(\mathcal{S},\mathrm{v},X=E)\ :\ E\subseteq U_{\mathcal{S}}\}$$
$$\llbracket \forall X.\phi\rrbracket(\mathcal{S},\mathrm{v}) = \sup\{\llbracket\phi\rrbracket(\mathcal{S},\mathrm{v},X=E)\ :\ E\subseteq U_{\mathcal{S}}\}$$

Asis the case for monadic logic, no property (if not trivial) is decidable for monadic logic in general. Since cost monadic logic is an extension of monadic logic, one cannot expect anything to be better in this framework. However we are interested, as in the standard setting, to decide properties over a restricted class $\mathcal{C}$ of structures. The class $\mathcal{C}$ can typically be the class of finite words, of finite trees, of infinite words (of length ω, or beyond) or of infinite trees. The subject of this paper is to consider the case of words over a fixed finite alphabet.

We are interested in deciding properties concerning the function described by cost monadic formulae over $\mathcal{C}$. But what kind of properties? It is quite easy to see that, given a cost monadic sentence ϕ and $n\in\infty$, one can effectively produce a monadic formula ϕ^n such that for all structures $\mathcal{S}$, $\mathcal{S}\models\phi^n$ iff $\llbracket\phi\rrbracket(\mathcal{S})=n$ (such a translation would be possible even if the positivity requirement in the use of the predicates $|X|\leq N$ is not required). Hence, deciding questions of the form '$\llbracket\phi\rrbracket=n$' can be reduced to the standard theory.

Properties that cannot be reduced to the standard theory, and that we are interested in, involve the existence of bounds. One says below that a function f is *bounded over* some set X if there is some integer n such that $f(x)\leq n$ for all $x\in X$. We are interested in the following generic problems:

Boundedness: Is the function $\llbracket\phi\rrbracket$ bounded over $\mathcal{C}$?
Or (variant), is $\llbracket\phi\rrbracket$ bounded over a regular subset of $\mathcal{C}$?
Or (limitedness), is $\llbracket\phi\rrbracket$ bounded over $\{\mathcal{S}\ :\ \llbracket\phi\rrbracket(\mathcal{S})\neq\infty\}$?

Divergence: For all n, do only finitely many $\mathcal{S}\in\mathcal{C}$ satisfy $\llbracket\phi\rrbracket(\mathcal{S})\leq n$?
Said differently, are all sets over which $\llbracket\phi\rrbracket$ is bounded of bounded cardinality?

Domination: For all $E\subseteq\mathcal{C}$, does $\llbracket\phi\rrbracket$ bounded over E imply $\llbracket\psi\rrbracket$ is also bounded over E?

All these questions cannot be reduced (at least simply) to questions in the standard theory. Furthermore, all these questions become undecidable for very standard reasons as soon as the requirement of positivity in the use of the new predicate $|X|\leq N$ is removed. In this paper, we introduce suitable material for proving their decidability over the class $\mathcal{C}$ of words.

One easily sees that the domination question is in fact a joint extension of the boundedness question (if one sets ϕ to be always true, i.e., to compute the constant function 0), and the divergence question (if one sets ψ to be measuring the size of the structure, i.e., $\forall X.|X|\leq N$). Let us remark finally that if ϕ is a formula of monadic logic, then the boundedness question corresponds to deciding if ϕ is a tautology. If furthermore ψ is also monadic, then the domination consists of deciding whether ϕ implies ψ.

In the following section, we introduce the notion of cost functions, i.e., equivalence classes over functions allowing to omit discrepancies of the function described, while preserving sufficient information for working with the above questions.

2.2. **Cost functions.**

In this section, we introduce the equivalence relation $\approx$ over functions, and the central notion of cost function.

A *correction function* α is a non-decreasing mapping from $\mathbb{N}$ to $\mathbb{N}$ such that $\alpha(n) \geq n$ for all n. From now on, the symbols α, $\alpha' \dots$ implicitly designate correction functions. Given x, y in $\mathbb{N} \cup \{\infty\}$, $x \preccurlyeq_\alpha y$ holds if $x \leq \overline{\alpha}(y)$ in which $\overline{\alpha}$ is the extension of α with $\overline{\alpha}(\infty) = \infty$. For every set E, $\preccurlyeq_\alpha$ is extended to $(\mathbb{N} \cup \{\infty\})^E$ in a natural way by $f \preccurlyeq_\alpha g$ if $f(x) \preccurlyeq_\alpha g(x)$ for all $x \in E$, or equivalently $f \leq \overline{\alpha} \circ g$. Intuitively, f is dominated by g after it has been 'stretched' by α. One also writes $f \approx_\alpha g$ if $f \preccurlyeq_\alpha g$ and $g \preccurlyeq_\alpha f$. Finally, one writes $f \preccurlyeq g$ (resp. $f \approx g$) if $f \preccurlyeq_\alpha g$ (resp. $f \approx_\alpha g$) for some α. A *cost function* (over a set E) is an equivalence class of $\approx$ (i.e., a set of mappings from E to $\mathbb{N} \cup \{\infty\}$).

Some elementary properties of $\preccurlyeq_\alpha$ are:

Fact 2.5. If $\alpha \leq \alpha'$ and $f \preccurlyeq_\alpha g$, then $f \preccurlyeq_{\alpha'} g$. If $f \preccurlyeq_\alpha g \preccurlyeq_{\alpha'} h$, then $f \preccurlyeq_{\alpha \circ \alpha'} h$.

The above fact allows to work with a single correction function at a time. Indeed, as soon as two correction functions α and α' are involved in the same proof, we can consider the correction function $\alpha'' = \max(\alpha, \alpha')$. By the above fact, it satisfies that $f \preccurlyeq_\alpha g$ implies $f \preccurlyeq_{\alpha''} g$, and $f \preccurlyeq_{\alpha'} g$ imples $f \preccurlyeq_{\alpha''} g$.

Example 2.6. Over $\mathbb{N} \times \mathbb{N}$, maximum and sum are equivalent for the doubling correction function (for short, $(\max) \approx_{\times 2} (+)$). Indeed, for all $x, y \in \omega$,

$$\max(x,y) \leq x + y \leq 2 \times \max(x,y) \ .$$

Our next examples concern mappings from sequences of words to $\mathbb{N}$. We have

$$|\ |_a \preccurlyeq |\ | \ , \qquad\qquad \text{and} \qquad |\ |_a \not\preccurlyeq |\ |_b \ ,$$

where a and b are distinct letters, and $|\ |$ maps a word to its length and $|\ |_a$ maps it to the number of occurrences of a. Indeed we have $|\ |_a \leq |\ |$ but the set of words a^* is a witness that $|\ |_a \preccurlyeq_\alpha |\ |_b$ cannot hold whatever is α.
Given words $u_1, \dots, u_k \in \{a,b\}^*$, we have

$$|u_1 \dots u_k|_a \approx_\alpha \max(|K|, \max_{i=1\dots k} |u_i|_a) \qquad \text{where} \quad K = \{i \in \{1,\dots,k\} \ : \ |u_i|_a \geq 1\}$$

in which α is the squaring function. Indeed, for one direction we just have to remark:

$$\max(|K|, \max_{i=1\dots k} |u_i|_a) \leq |u_1 \dots u_k|_a,$$

and for the other direction we use:

$$|u_1 \dots u_k|_a \leq \Sigma_{i \in K} |u_i|_a \leq (\max(|K|, \max_{i=1\dots k} |u_i|_a))^2 \ .$$

The relation $\preccurlyeq$ has other characterisations:

Proposition 2.1. For all f, g from E to $\mathbb{N} \cup \{\infty\}$, the following items are equivalent:
(1) $f \preccurlyeq g$,
(2) $\forall n \in \mathbb{N}. \exists m \in \mathbb{N}. \forall x \in E.\ g(x) \leq n \to f(x) \leq m$, and;

(3) for all $X \subseteq E$, $g|_X$ is bounded implies $f|_X$ is bounded.

Proof. From (1) to (2). Let us assume $f \preccurlyeq g$, i.e., $f \preccurlyeq_\alpha g$ for some α. Let n be some non-negative integer, and $m = \alpha(n)$. We have for all x, that $g(x) \leq n$ implies $f(x) \leq (\alpha \circ g)(x) \leq \alpha(n) = m$, thus establishing the second statement.

From (2) to (3). Let $X \subseteq E$ be such that $g|_X$ is bounded. Let n be a bound of g over X. Item (2) states the existence of m such that $\forall x \in E. g(x) \leq n \to f(x) \leq m$. In particular, for all $x \in E$, we have $g(x) \leq n$ by choice of n, and hence $f(x) \leq m$. Hence $f|_X$ is bounded by m.

From (3) to (1). Let $n \in \mathbb{N}$, consider the set $X_n = \{x \ : \ g(x) \leq n\}$. The mapping g is bounded over X_n (by n), and hence by (3), f is also bounded. We set $\alpha(n) = \sup f(X_n)$. Let now $x \in X$. If $g(x) < \infty$, we have that $x \in X_{g(x)}$ by definition of the X's. Hence $f(x) \leq \sup f(X_{g(x)}) = \alpha(g(x))$. Otherwise $g(x) = \infty$, and we have $f(x) \leq \overline{\alpha}(g(x)) = \infty$. Hence $f \preccurlyeq_\alpha g$. □

The last characterisation shows that the relation $\approx$ is an equivalence relation that preserves the existence of bounds. Indeed, all this theory can be seen as a method for proving the existence/non-existence of bounds. One can also remark that the questions of boundedness, divergence, and domination presented in the previous section, are preserved under replacing the semantic of a formula by an $\approx$-equivalent function. Furthermore, the domination question can be simply reformulated as $[\![\phi]\!] \succcurlyeq [\![\psi]\!]$.

Cost functions over some set E ordered by $\preccurlyeq$ form a lattice. Let us show how this lattice refines the lattice of subsets of E ordered by inclusion. The following elementary fact shows that we can identify a subset of E with the cost function of its characteristic function (given a subset $X \subseteq E$, one denotes by χ_X its *characteristic mapping* defined by $\chi_X(x) = 0$ if $x \in X$, and ∞ otherwise):

Fact 2.7. For all $X, Y \subseteq E$, $\chi_X \preccurlyeq \chi_Y$ iff $Y \subseteq X$.

In this respect, the lattice of cost functions is a refinement of the lattice of subsets of E equipped with the superset ordering. Let us show that this refinement is strict. Indeed, there is only one language L such that χ_L does not have ∞ in its range, namely $L = E$, however, we will show in Proposition 2.2 that, as soon as E is infinite, there are uncountably many cost functions which have this property of not using the value ∞.

Proposition 2.2. If E is infinite, then there exist at least continuum many different cost functions from E to $\mathbb{N}$.

Proof. Without loss of generality, let us assume that $E = \mathbb{N} \setminus \{0\}$. Let $p_0, p_1, \ldots$ be the sequence of all prime numbers. Every $n \in E$ is decomposed in a unique way as $p_1^{n_1} p_2^{n_2} \ldots$ in which all n_i's are null but finitely many (with an obvious meaning of the infinite product). For all $I \subseteq \mathbb{N}$, one defines the function f_I from $\mathbb{N} \setminus \{0\}$ to $\mathbb{N}$ for all $n \in \mathbb{N} \setminus \{0\}$ by:

$$f_I(n) = \max\{n_i \ : \ i \in I, \ n = p_1^{n_1} p_2^{n_2} \ldots\} \ .$$

Consider now two different sets $I, J \subseteq \mathbb{N}$. This means—up to a possible exchange of the roles of I and J—that there exists $i \in I \setminus J$. Consider now the set $X = \{p_i^k \ : \ k \in \mathbb{N}\}$. Then, by construction, $f_I(p_i^k) = k$ and hence f_I is not bounded over X. However, $f_J(p_i^k) = 0$ and hence f_J is bounded over X. It follows by Proposition 2.1 that f_I and f_J are not equivalent for $\approx$. We can finally conclude that—since there exist continuum many subsets of $\mathbb{N}$— there is at least continuum many cost functions over E which do not use value ∞. □

2.3. **Solving cost monadic logic over words using cost functions.** As usual, we see a word as a structure, the universe of which is the set of positions in the word (numbered from 1), equipped with the ordering relation $\leq$, and with a unary relation for each letter of the alphabet that we interpret as the set of positions at which the letter occur. Given a set of monadic variables F, and a valuation v of F over a word $u = a_1 \dots a_k \in \mathbb{A}^*$, we denote by $\langle u, \mathrm{v}\rangle$ the word $c_1 \dots c_k$ over the alphabet $\mathbb{A}_F = \mathbb{A} \times \{0,1\}^F$ such that for all position $i = 1 \dots k$, $c_i = (a_i, \delta_i)$ in which δ_i maps $X \in F$ to 1 if $i \in \mathrm{v}(X)$, and to 0 otherwise.

It is classical that given a formula ϕ with free variables F, the language

$$L_\phi = \{\langle u, \mathrm{v}\rangle \ :\ u, \mathrm{v} \models \phi\} \subseteq \mathbb{A}_F^*$$

is regular. The proof is done by induction on the formula. It amounts to remark that to the constructions of the logic, namely disjunction, conjunction, negation and existential quantification, correspond naturally some language theoretic operations, namely union, intersection, complementation and projection. The base cases are obtained by remarking that the relations of ordering, inclusion, and letter, also correspond to regular languages.

We use a similar approach. To each cost monadic formula ϕ with free variables F over the signature of words over $\mathbb{A}$, we associate the cost function f_ϕ over $\mathbb{A}_F$ defined by

$$f_\phi(\langle u, \mathrm{v}\rangle) = [\![\phi]\!](u, \mathrm{v}) \ .$$

We aim at solving cost monadic logic by providing an explicit representation to the cost functions f_ϕ. For reaching this goal, we need to define a family of cost functions $\mathcal{F}$ that contains suitable constants, has effective closure properties and decision procedures.

The first assumption we make is the closure under *composition with a morphism*. I.e., let f be a cost function in $\mathcal{F}$ over $\mathbb{A}^*$ and h be a morphism from $\mathbb{B}^*$ ($\mathbb{B}$ being another alphabet) to $\mathbb{A}^*$, we require $f \circ h$ to also belong to $\mathcal{F}$. In particular, this operation allows us to change the alphabet, and hence to add new variables when required. It corresponds to the closure under inverse morphism for regular languages.

Fact 2.4 gives us a very precise idea of the constants we need. The constants correspond to the formulae of the form $R(X_1, \dots, X_n)$ as well as their negation. As mentioned above, for such a formula ϕ, L_ϕ is regular. Hence, it is sufficient for us to require that the characteristic function χ_L belongs to $\mathcal{F}$ for each regular language L. The remaining constants correspond to the formula $|X| \leq N$. We have that $f_{|X|\leq N}(\langle u, X = E\rangle) = |E|$. This corresponds to counting the number of occurrences of $\mathbb{A} \times \{1\}$ in a word over $\mathbb{A} \times \{0,1\}$. Up to a change of alphabet (thanks to the closure under composition with a morphism) it will be sufficient for us that $\mathcal{F}$ contains the function "size" which maps each word $u \in \{a,b\}^*$ to $|u|_a$.

Fact 2.4 gives us also a very precise idea of the closure properties we need. We need the closure under min and max for disjunctions and conjunctions. For dealing with existential and universal quantification, we need the new operations of inf*-projection* and sup*-projection*. Given a mapping f from $\mathbb{A}^*$ to $\mathbb{N} \cup \{\infty\}$ and a mapping h from $\mathbb{A}$ to $\mathbb{B}$ that we extend into a morphism from $\mathbb{A}^*$ to $\mathbb{B}^*$ ($\mathbb{B}$ being another alphabet) the *inf-projection* of f with respect to h is the mapping $f_{\inf,h}$ from $\mathbb{B}^*$ to $\mathbb{N} \cup \{\infty\}$ defined for all $v \in \mathbb{B}^*$ by:

$$f_{\inf,h}(v) = \inf\{f(u) \ :\ h(u) = v\} \ .$$

Similarly, the *sup-projection* of f with respect to h is the mapping $f_{\sup,h}$ from $\mathbb{B}^*$ to $\mathbb{N} \cup \{\infty\}$ defined for all $v \in \mathbb{B}^*$ by:

$$f_{\sup,h}(v) = \sup\{f(u) \ :\ h(u) = v\} \ .$$

We summarise all the requirements in the following fact.

Fact 2.8. Let $\mathcal{F}$ be a class of cost functions over words such that:

(1) for all regular languages L, χ_L belongs to $\mathcal{F}$,
(2) $\mathcal{F}$ contains the cost function "size",
(3) $\mathcal{F}$ is effectively closed under composition with a morphism, min, max, inf-projection and sup-projection,
(4) $\preccurlyeq$ is decidable over $\mathcal{F}$,

then the boundedness, divergence and domination problems are decidable for cost monadic logic over words.

The remainder of the paper is devoted to the introduction of the class of recognisable cost functions, and showing that this class satisfies all the assumptions of Fact 2.8. In particular, Item 1 is established as Example 4.2. Item 2 is achieved in Example 4.1. Item 3 is the subject of Fact 4.3, Corollary 4.6 and Theorems 4.8 and 4.13. Finally, Item 4 is established in Theorem 4.7.

Thus we deduce our main result.

Theorem 2.3. The domination relation is decidable for cost-monadic logic over finite words.

All of those results are established in Section 4. However, we need first to introduce the notion of stabilisation monoids, as well as some of its key properties. This is the subject of Section 3.

3. The algebraic model: stabilisation monoids

The purpose of this section is to describe the algebraic model of stabilisation monoids. This model has – a priori – no relation with the previous section. However, in Section 4, in which we define the notion of a recognisable cost function, we will be able to put all of the pieces together.

The key idea – which is directly inspired from the work of Leung, Simon and Kirsten – is to develop an algebraic notion (the stabilisation monoid) in which a special operator (called the stabilisation, $\sharp$) allows to express what happens when one iterates "a lot of times" some element. In particular, it says whether one should count or not the number of iterations of this element. The terminology "a lot of times" is very vague, and for this reason such a formalism cannot describe precisely functions. However, it is perfectly suitable for describing cost functions.

The remaining part of the section is organised as follows. We first introduce the notion of stabilisation monoids in Section 3.1, paying a special attention to give it an intuitive meaning. In Section 3.2, we introduce the key notions of computations, under-computations and over-computations, as well as the two central results of existence of computations (Theorem 3.3) and "unicity" of their values (Theorem 3.4). Those notions and results form the main technical core of this work. Then Section 3.4 is devoted to the proof of Theorem 3.3, and Section 3.5 the proof of Theorem 3.4.

3.1. Stabilisation monoids.

A *semigroup* $\mathbf{S} = \langle S, \cdot \rangle$ is a set S equipped with an associative operation '$\cdot$'. A *monoid* is a semigroup such that the product has a *neutral element* 1, *i.e.*, such that $1 \cdot x = x \cdot 1 = x$ for all $x \in S$. Given a semigroup $\mathbf{S} = \langle S, \cdot \rangle$, one extends the product to products of arbitrary length by defining π from S^+ to S by $\pi(a) = a$

and $\pi(ua) = \pi(u) \cdot a$. If the semigroup is a monoid of neutral element 1, one further set $\pi(\varepsilon) = 1$. All semigroups are monoids, and conversely it is sometimes convenient to transform a semigroup $\mathbf{S}$ into a monoid $\mathbf{S}^1$ simply by the adjunction of a new neutral element 1.

An idempotent in $\mathbf{S}$ is an element $e \in S$ such that $e \cdot e = e$. One denotes by $E(\mathbf{S})$ the set of idempotents in $\mathbf{S}$. An *ordered semigroup* $\langle S, \cdot, \leq \rangle$ is a semigroup $\langle S, \cdot \rangle$ together with an order $\leq$ over S such that the product $\cdot$ is *compatible* with $\leq$; *i.e.*, $a \leq a'$ and $b \leq b'$ implies $a \cdot b \leq a' \cdot b'$. An *ordered monoid* is an ordered semigroup, the underlying semigroup of which is a monoid.

We are now ready to introduce the new notion of stabilisation semigroups and stabilisation monoids.

Definition 3.1. A *stabilisation semigroup* $\langle S, \cdot, \leq, \sharp \rangle$ is a *finite* ordered semigroup $\langle S, \cdot, \leq \rangle$ together with an operator $\sharp \colon E(\mathbf{S}) \to E(\mathbf{S})$ (called the *stabilisation*) such that:

- for all $e \leq f$ in $E(\mathbf{S})$, $e^\sharp \leq f^\sharp$;
- for all $a, b \in S$ with $a \cdot b \in E(\mathbf{S})$ and $b \cdot a \in E(\mathbf{S})$, $(a \cdot b)^\sharp = a \cdot (b \cdot a)^\sharp \cdot b$;[3]
- for all $e \in E(\mathbf{S})$, $e^\sharp \leq e$;
- for all $e \in E(\mathbf{S})$, $(e^\sharp)^\sharp = e^\sharp$.

It is called a *stabilisation monoid* if furthermore $\langle S, \cdot \rangle$ is a monoid and $1^\sharp = 1$ in which 1 is the neutral element of the monoid.

The intuition is that $e^\sharp$ represents what is the value of e^n when n becomes "very large". Some consequences of the definitions, namely[4]

$$\text{for all } e \in E(\mathbf{S}), \qquad e^\sharp = e \cdot e^\sharp = e^\sharp \cdot e = e^\sharp \cdot e^\sharp = (e^\sharp)^\sharp \ ,$$

make perfect sense in this respect: repeating "a lot of e's" is equivalent to seeing one e followed by "a lot of e's", etc... This meaning of $e^\sharp$ is in some sense a limit behaviour. This is an intuitive reason why $\sharp$ is not used for non-idempotent elements. Consider for instance the element 1 in $\mathbb{Z}/2\mathbb{Z}$. Then iterating it yields 0 at even iterations, and 1 at odd ones. This alternation prevents to giving a clear meaning to what is the result of "iterating a lot of times".

However, this view is incompatible with the classical view on monoids, in which by induction, if $e \cdot e = e$, then $e^n = e$ for all $n \geq 1$. The idea in stabilisation monoids is that the product is something that cannot be iterated "a lot of times". This implies that thinking that for all $n \geq 1$, e^n is the same as e becomes something "incorrect" for "large" values of n. The value of e^n is e if n is "small", and it is $e^\sharp$ if n is "big". Most of the remainder of the section is devoted to the formalisation of this intuition, via the use of the notion of computations.

Even if the material necessary for working with stabilisation monoids has not been yet provided, it is already possible to give some examples of stabilisation monoids that are constructed from an informal idea of their intended meaning.

Example 3.1. In this example we start from an informal idea of what we would like to compute, and construct a stabilisation monoid from it. The explanations have to remain

[3]This equation states that $\sharp$ is a *consistent mapping* in the sense of [26, 28].

[4]Indeed, $e^\sharp = (e \cdot 1)^\sharp = e^{\cdot}(1 \cdot e)^\sharp \cdot 1 = e \cdot e^\sharp$ using consistency. In the same way $e^\sharp = e^\sharp \cdot e$. Since $\sharp$ maps idempotents to idempotents, $e^\sharp = e^\sharp \cdot e^\sharp$ is obvious, and $(e^\sharp)^\sharp$ is also by definition.

informal at this point in the exposition of the theory. However, this example should illustrate how one can already reason easily at this level of understanding.

Imagine you want, among words over a and b, to separate the ones that possess 'a lot of occurrences of a's' from the ones that have only 'a few occurrences of a's', i.e., imagine you want to describe a stabilisation monoid which would "count" the number of occurrences of a's.

For doing this, one should separate three "kinds" of words:

- the kind of words with no occurrence of a; let b be the corresponding element in the stabilisation monoid (since the word b is of this kind),
- the kind of words with at least one occurrence of a, but only "a few" such occurrences; let a be the corresponding element in the stabilisation monoid (since the word a is of this kind),
- the kind of words with "a lot of occurrences of a's"; let 0 be the corresponding element in the stabilisation monoid.

The words that we intend to separate – the ones with a lot of a's – are the ones of kind 0. With these three elements known, let us complete the definition of the stabilisation monoid.

Of course, iterating twice, or "many times", words which contain no occurrences of a yields words with no occurrence of the letter a. We capture this with the equalities $b = b \cdot b = b^\sharp$.

Now words that have at least one a, but only a few number of occurrences of a's, should not be affected by appending b letters to their left or to their right. I.e., we set $a = b{\cdot}a = a{\cdot}b$. Even more, appending a word with 'few a's' to another word with 'few a's' does also give a word with "few a's". I.e., we set $a \cdot a = a$.

However, if we iterate 'a lot of times' a word with at least one occurrence of a, it yields a word with "a lot of a's". Hence we set $a^\sharp = 0$. We also easily get the equations $b \cdot 0 = 0 \cdot b = a \cdot 0 = 0 \cdot a = 0 \cdot 0 = 0^\sharp = 0$ by inspecting all situations. We reach the following description of the stabilisation monoid:

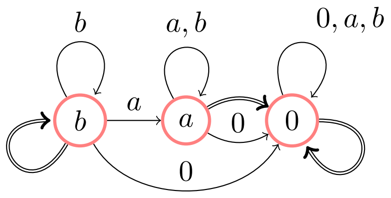
	b	a	0	$\sharp$
b	b	a	0	b
a	a	a	0	0
0	0	0	0	0

The left part describes the product operation $\cdot$, while the rightmost column gives the value of stabilisation $\sharp$ (in general, this column may be partially defined since stabilisation is defined only for idempotents).

To complete the definition, we need to define the ordering over $\{b, a, 0\}$. The least we can do is setting $0 \leq a$ and $x \leq x$ for all $x \in \{b, a, 0\}$. This is mandatory, since by definition of a stabilisation monoid $a^\sharp \leq a$, and we have $a^\sharp = 0$. The reader can check that all the properties that we expect from a stabilisation monoid are now satisfied.

The intuition behind the ordering is that depending on what we mean by "a lot of a's", the same word can be of kind a or of kind 0. For instance the word a^{100} is of kind a if we consider that 100 is "a few", while it is of kind 0 if we consider that 100 is "a lot". For this reason, there is a form of continuum that allows to go from a to 0. The order $\leq$ captures this relationship between elements.

It is sometime convenient to present a stabilisation monoid by a form of Cayley graph:



As in a standard Cayley graph, there is an edge labeled by y going from every vertex x to $x \cdot y$. Furthermore, there is a double arrow linking every idempotent x to its stabilised version $x^\sharp$.

Example 3.2. Imagine one wants to compute the size of the longest sequence of consecutive a's in words over the alphabet $\{a, b\}$. Then we would separate four "kinds" of words:

- the kind consisting only of the empty word; let it be 1,
- the kind of words, containing only occurrences of a, at least one occurrence of it, but only "a few" of them; let the corresponding element be a,
- the kind of words containing at least one b, but no long sequence of consecutive a's; let the corresponding element be b,
- the kind of words that contain a long sequence of consecutive a's; let the corresponding element be 0.

Computing the size of the longest sequence of consecutive a's means identifying the words containing a "long" sequence of this type, *i.e.*, it means to separate words of kind 0 from words of kind a or b.

The table of product and stabilisation is then naturally the following:

	1	a	b	0	$\sharp$
1	1	a	b	0	1
a	a	a	b	0	0
b	b	b	b	0	b
0	0	0	0	0	0

We complete the definition of this stabilisation monoid by defining the ordering. For this, we let $x \leq x$ hold for any $x \in \{1, a, b, 0\}$, and we further set $0 \leq a$ since $a^\sharp = 0$. Since $0 = 0 \cdot b$, $0 \leq a$ and $a \cdot b = b$, we need also to set $0 \leq b$ for ensuring the compatibility of the product with the order. Once more the ordering corresponds to the intuition that there exist words that can be of kind a (e.g., the word a^{100}) or b (e.g., the word ba^{100}), and that have kind 0 if we change what we mean by "a lot of".

Remark 3.3. The notion of stabilisation monoids (or semigroups) extends the one of standard monoids (or semigroups). Many standard results concerning monoids have natural counterparts in the world of stabilisation monoids. For making this relationship more precise, let us describe the canonical way to translate a monoid into a stabilisation monoid.

Let $\mathbf{M} = \langle M, \cdot \rangle$ be a monoid. The corresponding stabilisation monoid is:

$$\mathbf{M}_\sharp = \langle M, \cdot, =, id_{E(M)} \rangle .$$

In other words, the monoid is extended with a trivial ordering (the equality), and the stabilisation is simply the identity over idempotents. The reader can easily check that this object indeed respects the definition of a stabilisation monoid.

If we refer to the intuition we gave above, one extends the monoid by an identity stabilisation. This means that for all idempotents, one does not make the distinction between iterating it “a few times”, and “a lot of times”. Said differently, one never has to count the number of occurrences of the idempotents. This is consistent with the principle that a standard monoid has no counting capabilities.

Remark 3.4. The order plays an important role, even if it is sometimes hidden. Let us first remark that given a stabilisation monoid, it may happen that changing the order yields again another valid stabilisation monoid (as for ordered monoids). In general, there is a least order such that the structure is a valid stabilisation monoid. It is the intersection of all the “valid orders”, and can be computed by a least fix-point. However, there is no maximal “valid order” in general.

More interestingly, there exist structures $\langle M, \cdot, \sharp\rangle$ which have no order, which satisfy the definition of a stabilisation monoid except the rules involving the order, and such that it is not possible to construct an order for making them a valid stabilisation monoid. An example is the 10 element structure which would be obtained for describing the property “there is an even number of long maximal segments of a’s”. But this is what we want. Indeed, a closer inspection would reveal that this property does not have the monotonic behaviour that we could use for defining a function. Consider for instance a word of the form $a^1ba^2ba^3b\dots ba^n$, then shifting the threshold of what is considered to be large would yield a non monotonic behaviour, where even values would satisfy the property, and odd ones would not. Keeping in mind cost-monadic logic from the previous section, one sees that no formula would be able to express such a property. Requiring an order in the definition of stabilisation monoids rules out such situations.

We have seen through the above examples how easy it is to work with stabilisation monoids at an informal level. An important part of the remainder of the section is dedicated to providing formal definitions for this informal reasoning. In the above explanations, we worked with the imprecise terminology “a few” and “a lot of”. Of course, the value (what we referred to as “the kind” in the examples) of a word depends on what is the frontier we fix for separating “a few” from “a lot”.

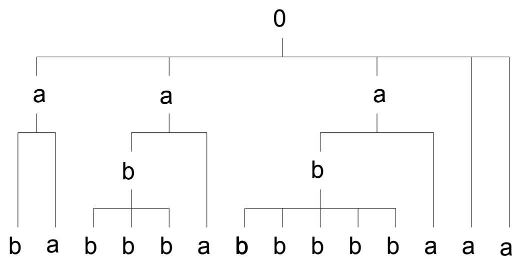
We continue the description of stabilisation monoids by introducing the key notion of computations. These objects describe how to evaluate a long “product” in a stabilisation monoid.

3.2. **Computations, under-computations and over-computations.**

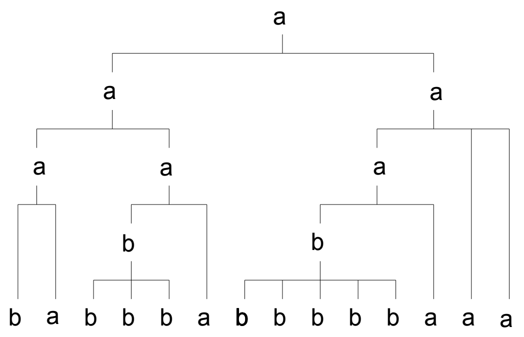
Our goal is now to provide a formal meaning for the notion of stabilisation semigroups and stabilisation monoids, allowing to avoid terms such as “a lot” or “a few”. More precisely, we develop in this section the notion of computations. A computation is a tree which is used as a witness that a word evaluates to a given value.

We fix ourselves for the rest of the section a stabilisation semigroup $\mathbf{S} = \langle S, \cdot, \sharp, \leq\rangle$. We develop first the notion for semigroups, and then see how to use it for monoids in Section 3.3 (we will see that the notions are in close correspondence).

Let us consider a word $u \in S^+$ (it is a word over S, seen as an alphabet). Our objective is to define a “value” for this word. In standard semigroups, the “value” of u is simply $\pi(u)$, the product of the elements appearing in the word. But, what should the “value” be for a stabilisation semigroup? All the informal semantics we have seen so far were based on the distinction between “a few” and “a lot”. This means that the value the word has depends on what is considered as “a few”, and what is considered as “a lot”. This is captured by the fact that the value is parameterised by a positive integer n which can be understood



A 4-computation of value 0 and height 4



A 4-computation of value a and height 5

Figure 1: Two 4-computations for the word *babbbabbbbbaaa* in the monoid of Example 3.1.

as a *threshold* separating what is considered as "a few" from what is considered as "a lot". For each choice of n, the word u is subject to have a different value in the stabilisation semigroup.

Let us assume a threshold value n is fixed. We still lack a general mechanism for associating to each word u over S a value in S. This is the purpose of *computations*. Computations are proofs (taking the form of a tree) that a word should evaluate to a given value. Indeed, in the case of usual semigroups, the fact that a word u evaluates to $\pi(u)$ can be witnessed by a binary tree, the leaves of which read from left to right yield the word u, and such that each inner node is labelled by the product of the label of its children. Clearly, the root of such a tree is labelled by $\pi(u)$, and the tree can be seen as a proof of correctness for this value.

The notion of n-computation that we define now is a variation around this principle. For more ease in its use, it comes in three variants: under-computations, over-computations and computations.

Definition 3.2. An *n-under-computation T for the word $u = a_1 \dots a_l \in S^+$* is an ordered unranked tree with l leaves, each node x of which is labelled by an element $v(x) \in S$ called the *value of* x, and such that for each node x of children $y_1, \dots, y_k$ (read from left to right), one of the following cases holds:

Leaf: $k = 0$, and $v(x) \leq a_m$ where x is the mth leave of T (read from left to right),
Binary node: $k = 2$, and $v(x) \leq v(y_1) \cdot v(y_2)$,

Idempotent node: $2 \leq k \leq n$ and $v(x) \leq e$ where $e = v(y_1) = \cdots = v(y_k) \in E(M)$,
Stabilisation node: $k > n$ and $v(x) \leq e^\sharp$ where $e = v(y_1) = \cdots = v(y_k) \in E(M)$.

An *n-over computation* is obtained by replacing everywhere "$v(x) \leq$" by "$v(x) \geq$". An *n-computation* is obtained by replacing everywhere "$v(x) \leq$" by "$v(x) =$", *i.e.*, a n-computation is a tree which is at the same time an n-under computation and an n-over computation.

The *value* of a [under-/over-]computation is the value of its root.

We also use the following notations for easily denoting constructions of [under/over]-computations. Given a non-leaf S-labelled tree T, denote by T^i the subtree of T rooted at the i^{th} children of the root. For a in S, one denotes as a the tree restricted to a single leaf of value a. If furthermore $T_1, \ldots, T_k$ are also S-labelled trees, then $a[T_1, \ldots, T_k]$ denotes the tree of root labelled a, of degree k, and such that $T^i = T_i$ for all $i = 1 \ldots k$.

It should be immediately clear that those notions have to be manipulated with care, as shown by the following example.

Example 3.5. Two examples of computations are given Figure 1. Both correspond to the stabilisation semigroup (in fact monoid) of Example 3.1, the aim of which is to count the number of occurrences of the letter a in a word. Both correspond to the evaluation of the same word. Both correspond to the same threshold value n. However, those two computations do not have the same value. We will see below how to compare computations and overcome this problem.

There is another problem. Indeed, it is straightforward to construct an n-computation for some word, simply by constructing a computation which is a binary tree, and would use no idempotent nodes nor stabilisation nodes. However, such a computation would of course not be satisfactory since every word would be evaluated in this way as if the stabilisation semigroup was a standard semigroup, and we do not want that. We need to determine what is a relevant computation in order to rule out such computations.

Thus we need to answer the following questions:

(1) What are the relevant computations?
(2) Can we construct a relevant n-computation for all words and all n?
(3) How can we relate the different values that n-computations may have on the same word?

The answer to the first question is that we are only interested in computations of small height, meaning of height bounded by some function of the semigroup. With such a restriction, it is not possible to use binary trees as computations. However, this choice makes the answer to the second question less obvious: does there always exist a computation?

Theorem 3.3. For all words $u \in S^+$ and all non-negative integers n, there exists an n-computation of height at most[5] $3|S|$.

This result is an extension of the forest factorisation theorem of Simon [42] (which corresponds to the case of a semigroup). Its proof, which is independent from the rest of this work, is presented in Section 3.4.

The third question remains: how to compare the values of different computation over the same word? An answer to this question in its full generality makes use of under- and over-computations.

[5]When measuring the height of a tree, the convention is that leaves do not count. With this convention, substituting a tree for a leaf of another tree results in a tree of height the sum of the heights of the two trees.

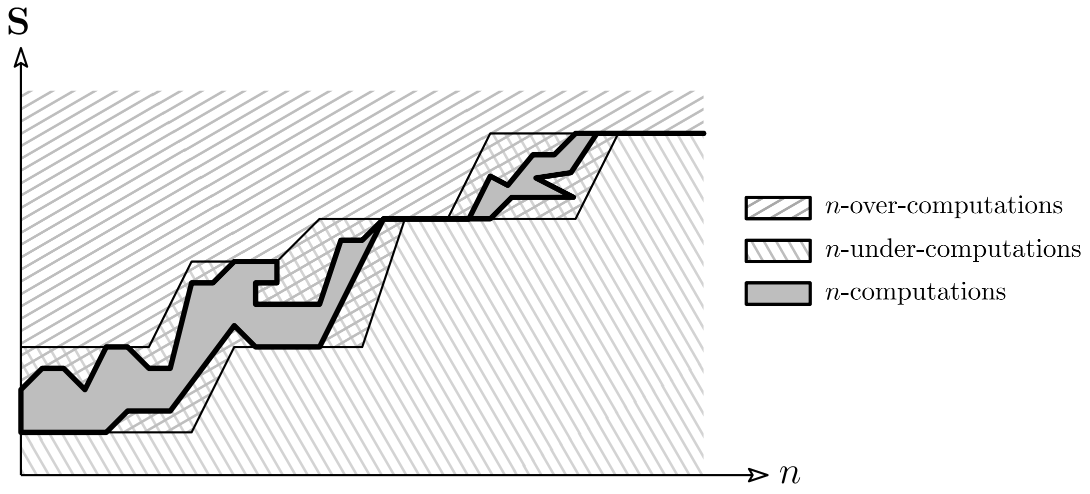


Figure 2: Structure of computations, under-computations, and over-computations.

Theorem 3.4. For all non-negative integers p, there exists a polynomial $\alpha : \mathbb{N} \to \mathbb{N}$ such that for all n-under-computations over some word of value a and height at most p, and all $\alpha(n)$-over computations of value b over the same word u,

$$a \leq b \; .$$

Remark first that since computations are special instances of under- and over-computations, Theorem 3.4 holds in particular for comparing the values of computations. The proof of Theorem 3.4 is the subject of Section 3.5.

We have illustrated the above results in Figure 2. It depicts the relationship between computations in some idealised stabilisation monoid $\mathbf{S}$. In this drawing, assume some word over some stabilisation semigroup is fixed, as well as some integer $p \geq 3|\mathbf{S}|$. One aims at representing for each n the possible values of an n-computation, an n-under computation or an n-over computation for this word of height at most p. In all the explanations below, all computations are supposed to not exceed height p.

The horizontal axis represents the n-coordinate. The values in the stabilisation semigroup being ordered, the vertical axis represents the values in the stabilisation semigroup (for the picture, we assume the values in the stabilisation semigroup totally ordered). Thus an n-computation (or n-under or n-over-computation) is placed at a point of horizontal coordinate n and vertical coordinate the value of the computation.

We can now interpret the properties of the computations in terms of this figure. First of all, under-computations as well as over-computations, and as opposed to computations, enjoy certain forms of monotonicity as shown by the fact below.

Fact 3.6. For $m \leq n$, all m-under-computations are also n-under-computations, and all n-over-computations are also m-over-computations (using the fact that $e^\sharp \leq e$ for all idempotents e).

Any n-under-computation of value a can be turned into an n-under-computation of value b for all $b \leq a$ (by changing the root label from a to b). Similarly any n-over-computation of value a can be turned into an n-over-computation of value b for all $b \geq a$.

Fact 3.6 is illustrated by Figure 2. It means that over-computations define a left and upward-closed area, while the under-computations define a right and downward-closed area. Hence, in particular, the delimiting lines are non-decreasing. Furthermore, since computations are at the same-time over-computations and under-computations, the area of computations lie inside the intersection of under-computations and over-computations. Since the height p is chosen to be at least $3|\mathbf{S}|$, Theorem 3.3 provides for us even more information. Namely, for each value of n, there exists an n-computation. This means in the picture that the area of computations crosses every column. However, since computations do not enjoy monotonicity properties, the shape of the area of computations can be quite complicated. Finally Theorem 3.4 states that the frontier of under-computations and the frontier of over-computations are not far one from each other. More precisely, if one chooses an element a of the stabilisation semigroup, and one draws an horizontal line at altitude a, if the frontier of under-computations is above or at a for threshold n, then the frontier of over-computations is also above or at a at threshold $\alpha(n)$. Hence the frontier of over-computations is always below the one of under-computations, but it essentially grows at the same speed, with a delay of at most α.

Remark 3.7. In the case of a standard semigroups or monoids (which can be seen as stabilisation monoids or semigroups according to Remark 3.3), the notions of computations, under-computations and over-computations coincide (since the order is trivial), and the value of the threshold n becomes also irrelevant. This means that the value of all n-[under/over-]computations over a word u coincide with $\pi(u)$. (Such computations are referred to as "Ramsey factorisations" in the factorisation forest theorem.)

Let us finally remark that Theorem 3.4, which is a consequence of the axioms of stabilisation semigroups, is also sufficient for deducing them. This is formalised by the following proposition.

Proposition 3.5. Let $\mathbf{S} = \langle S, \cdot, \leq, \sharp\rangle)$ be a a structure consisting of a finite set S, a binary operation $\cdot$ from S^2 to S, $\leq$ be a partial order, and $\sharp$ from S to S be a mapping defined over the idempotents of S. Assume furthermore that there exists α such that for all $\leq n$-under-computations for some word u of value a of height at most 3 and all $\geq \alpha(n)$-over-computation over u of value b of height at most 3,

$$a \leq b\ .$$

Then $\mathbf{S}$ is a stabilisation semigroup.

Proof. Let us first prove that $\cdot$ is compatible with $\leq$. Assume $a \leq a'$ and $b \leq b'$, then $(a \cdot b)[a, b]$ is a 0-under-computation over ab and $(a' \cdot b')[a', b']$ is an $\alpha(0)$-over-computation over the same word ab. It follows that $a \cdot b \leq a' \cdot b'$.

Let us now prove that $\cdot$ is associative. Let a, b, c in n. Then $((a \cdot b) \cdot c)[(a \cdot b)[a, b], c]$ is a 0-computation for the word abc, and $(a \cdot (b \cdot c))[a, (b \cdot c)[b, c]]$ is an $\alpha(0)$-computation for the same word. It follows that $(a \cdot b) \cdot c \leq a \cdot (b \cdot c)$. The other inequality is symmetric.

Let e be an idempotent. The tree $e^\sharp[\overbrace{e, \dots, e}^{2\alpha(0)+2}]$ is both a 0 and $\alpha(0)$-computation over the word $e^{2\alpha(0)+1}$. Furthermore, the tree $(e^\sharp \cdot e^\sharp)[e^\sharp[\overbrace{e, \dots, e}^{\alpha(0)+1}], e^\sharp[\overbrace{e, \dots, e}^{\alpha(0)+1}]]$ is also both a 0 and an $\alpha(0)$-computation for the same word. It follows that $e^\sharp \cdot e^\sharp = e^\sharp$, i.e., that $\sharp$ maps idempotents to idempotents.

Let us show that $e^\sharp \leq e$ for all idempotents e. The tree $e^\sharp[e,e,e]$ is a 2-computation over the word eee, and $e[e,e,e]$ is a $max(3,\alpha(2))$-computation over the same word. It follows that $e^\sharp \leq e$.

Let us show that stabilisation is compatible with the order. Let $e \leq f$ be idempotents. Then $e^\sharp[\overbrace{e,\dots,e}^{\alpha(0)+1}]$ and $f^\sharp[\overbrace{f,\dots,f}^{\alpha(0)+1}]$ are respectively a 0-computation for the word $e^{\alpha(0)+1}$ and an $\alpha(0)$-over-computation for the same word. It follows that $e^\sharp \leq f^\sharp$.

Let us prove that stabilisation is idempotent. Let e be an idempotent. We already know that $(e^\sharp)^\sharp \leq e^\sharp$. Let us prove the opposite inequality. Consider the 0-computation $e^\sharp[\overbrace{e,\dots,e}^{(\alpha(0)+1)^2}]$ for the word $e^{(\alpha(0)+1)^2}$, and the $\alpha(0)$-computation $(e^\sharp)^\sharp[\overbrace{e^\sharp[\overbrace{e,\dots,e}^{\alpha(0)+1}],\dots,e^\sharp[\overbrace{e,\dots,e}^{\alpha(0)+1}]}^{\alpha(0)+1}]$ for the same word. It follows that $e^\sharp \leq (e^\sharp)^\sharp$.

Let us finally prove the consistency of stabilisation. Assume that both $a \cdot b$ and $b \cdot a$ are idempotents. Let t_{ab} be $(a\cdot b)[a,b]$ (and similarly for t_{ba}), i.e., computations for ab and ba respectively. Define now:

$$t_{(a\cdot b)^\sharp} = (a\cdot b)^\sharp[\overbrace{t_{ab},\dots,t_{ab}}^{\alpha(0)+2\text{ times}}]\ ,$$

$$\text{and}\qquad t_{a\cdot(b\cdot a)^\sharp\cdot b} = (a\cdot(b\cdot a)^\sharp\cdot b)[a,((b\cdot a)^\sharp\cdot b)(b\cdot a)^\sharp[\overbrace{t_{ba},\dots,t_{ba}}^{\alpha(0)+1\text{ times}}],b]]\ .$$

Then both $t_{(a\cdot b)^\sharp}$ and $t_{a\cdot(b\cdot a)^\sharp\cdot b}$ are at the same time 0 and $\alpha(0)$-computations over the same word $(ab)^{\alpha(0)+2}$ of height at most 3. Since their respective values are $(a\cdot b)^\sharp$ and $a\cdot(b\cdot a)^\sharp\cdot b$, it follows by our assumption that $(a\cdot b)^\sharp = a\cdot(b\cdot a)^\sharp\cdot b$. □

This result is particularly useful. Indeed, when constructing a new stabilisation semigroup, one usually aims at proving that it 'recognises' some function. And this involves proving the hypothesis of Proposition 3.5. Thanks to Proposition 3.5, the syntactic correctness is then for free. This situation occurs in particular in Section 4.5 and 4.6 when the closure of recognisable cost-functions under inf-projection and sup-projection is established.

3.3. Specificities of stabilisation monoids.

We have presented so far the notion of computations in the case of stabilisation semigroups. We are in fact interested in the study of stabilisation monoids. Monoids differ from semigroups in the presence of a unit element 1. This element is used for modelling the empty word. We present in this section the natural variant of the notions of computations for the case of stabilisation monoids. As is often the case, results from stabilisation semigroups transfer naturally to stabilisation monoids. The definition is highly related to the one for stabilisation semigroups, and we see through this section that it is easy to go from the notion for stabilisation monoid to the one of stabilisation semigroup case, and backward. The result is that we use the same name "computation" for the two notions elsewhere in the paper.

Definition 3.6. Let $\mathbf{M}$ be a stabilisation monoid. Given a word $u \in M^*$, a *stabilisation monoid n-[under/over]-computation* (*sm*-[under/over]-computation for short) for u is an

n-[under/over]-computation for some $v \in M^+$, such that it is possible to obtain u from v by deleting some occurrences of the letter 1. All have value 1.

Thus, the definition deals with the implicit presence of arbitrary many copies of the empty word (the unit) interleaved with a given word. This definition allows us to work in a transparent way with the empty word (this saves us case distinctions in proofs). In particular the empty word has an sm-n-computation which is simply 1, of value 1. There are many others, like $1[1, 1, 1, 1[1, 1]]$ for instance.

Since each n-computation is also an sm-n-computation over the same word, it is clear that Theorem 3.3 can be extended to this situation (just the obvious case of the empty word need be treated separately):

Fact 3.8. There exists an sm-n-computation for all words in M^* of size at most $3|M|$.

The following lemma shows that sm-[under/over]-computations are not more expressive than [under/over]-computations. It is also elementary to prove.

Lemma 3.7. Given an sm-n-computation (*resp.*, sm-n-under-computation, sm-n-over-computation) of value a for the empty word, then $a = 1$ (*resp.* $a \leq 1$, $a \geq 1$).

For all non-empty words u and all sm-n-computations T (*resp.*, sm-n-under-computations, sm-n-over-computations) for u of value a, there exists an n-computation (*resp.*, n-under-computations, n-over-computations) for u of value a. Furthermore, its height is at most the height of T.

Proof. It is simple to eliminate each occurrence of an extra 1 by local modifications of the structure of the sm-computation: replace subtrees of the form $1[1, \ldots, 1]$ by 1, subtrees of the form $a[T, 1]$ by T, and subtrees of the form $a[1, T]$ by T, up to elimination of all occurrences of 1. For the empty word, this results in the first part of the lemma. For non-empty words, the resulting simplified sm-computation is a computation. The argument works identically for the under/over variants. □

A corollary is that Theorem 3.4 extends to sm-computations.

Corollary 3.8. For all non-negative integers p, there exists a polynomial $\alpha : \mathbb{N} \to \mathbb{N}$ such that for all sm-n-under-computations over some word $u \in M^*$ of value a and height at most p, and all sm-$\alpha(n)$-over computations of value b over the same word u,

$$a \leq b\ .$$

Proof. Indeed, the sm-under-computations and sm-over-computations can be turned into under-computations and over-computations of the same respective values by Lemma 3.7. The inequality holds for these under and over-computations by Theorem 3.4. □

There is a last lemma which is related and will prove useful.

Lemma 3.9. Let u be a word in M^* and v be obtained from u by eliminating some of its 1 letters, then all n-[under/over]-computations for v can be turned into an n-[under/over]-computation for u of same value. Furthermore, the height increase is at most 3.

Proof. Let $v = a_1 \ldots a_n$, then $u = u_1 \ldots u_n$ for $u_i \in 1^* a_i 1^*$. Let T be the n-[under/over]-computation for v of value a. It is easy to construct an n-[under/over]-computation T_i for u_i of height at most 3 of value a_i. It is then sufficient to plug in T each T_i for the ith leave of T. □

The consequence of these results is that we can work with sm-[under/over]-computations as with [under-over]-computations. For this reason we shall not distinguish further between the two notions unless necessary.

3.4. **Existence of computations: proof of Theorem 3.3.**

In this section, we establish Theorem 3.3 which states that for all words u over a stabilisation semigroup $\mathbf{S}$ and all non-negative integers n, there exists an n-computation for u of height at most $3|S|$. Remark that the convention in this context is to measure the height of a tree without counting the leaves. This result is a form of extension of the factorisation forest theorem due to Simon [42]:

Theorem 3.10 (Simon [42, 44])**.** Define a *Ramsey factorisation* to be an n-computation in the pathological case $n = \infty$ (i.e., there are no stabilisation nodes, and idempotent nodes are allowed to have arbitrary degree).
For all non-empty words u over a finite semigroup $\mathbf{S}$, there exists a Ramsey factorisation for u of height[6] at most $3|\mathbf{S}| - 1$.

Some proofs of the factorisation forest theorem can be found in [30, 7, 8]. Our proof could follow similar lines as the above one. Instead of that, we try to reuse as much lemmas as possible from the above constructions.

For proving Theorem 3.3, we will need one of Green's relations, namely the $\mathcal{J}$-relation (while there are five relations in general). Let us fix ourselves a semigroup $\mathbf{S}$. One denotes by $\mathbf{S}^1$ the semigroup extended (if necessary) with a neutral element 1 (this transforms $\mathbf{S}$ into a monoid). Given two elements $a, b \in S$, $a \leq_{\mathcal{J}} b$ if $a = x \cdot b \cdot y$ for some $x, y \in S^1$. If $a \leq_{\mathcal{J}} b$ and $b \leq_{\mathcal{J}} b$, then $a\mathcal{J}b$. We write $a <_{\mathcal{J}} b$ to denote $a \leq_{\mathcal{J}} b$ and $b \not\leq_{\mathcal{J}} a$. The interested reader can see e.g., [8] for an introduction to the relations of Green (with a proof of the original factorisation forest theorem), or monographs such as [31] , [15] or [39] for deep presentations of this theory. Finally, let us call a *regular element* in a semigroup an element a such that $a \cdot x \cdot a = a$ for some $x \in S^1$.

Lemma 3.11. Given a $\mathcal{J}$-class J in a finite semigroup, the following facts are equivalent:

- J contains an idempotent,
- J contains a regular element,
- there exist $a, b \in J$ such that $a \cdot b \in J$,
- all elements in J are regular,
- all elements in J can be written as $e \cdot c$ for some idempotent $e \in J$,
- all elements in J can be written as $c \cdot e$ for some idempotent $e \in J$.

Such $\mathcal{J}$-classes are called *regular*.

We will use the following technical lemma.

Lemma 3.12. If $f = e \cdot x \cdot e$ for $e\mathcal{J}f$ two idempotents, then $e = f$.

Proof. We use some standard results concerning finite monoids (without further explanations). The interested reader can find e.g., in [39] the necessary material. Since $e \cdot x \cdot e = f$, $f \leq_{\mathcal{L}} e$. Since furthermore $f\mathcal{J}e$, we have $f\mathcal{L}e$. In the same way $f\mathcal{R}e$. Overall $f\mathcal{H}e$. Since an $\mathcal{H}$-class contains at most one idempotent, $f = e$. □

[6]The exact bound of $3|S| - 1$ is due to Kufleitner [30]. It is likely that the same bound could be achieved for Theorem 3.3. We prefer here a simpler proof with a bound of $3|S|$.

The next lemma shows that the stabilisation operation behaves in a very uniform way inside $\mathcal{J}$-classes (similar arguments can be found in the works of Leung, Simon and Kirsten).

Lemma 3.13. *If $e\mathcal{J}f$ are idempotents, then $e^\sharp\mathcal{J}f^\sharp$. Furthermore, if $e = x \cdot f \cdot y$ for some x, y, then $e^\sharp = x \cdot f^\sharp \cdot y$.*

Proof. For the second part, assume $e = x \cdot f \cdot y$ and $e\mathcal{J}f$. Let $f' = (f \cdot y \cdot e \cdot x \cdot f)$. One easily checks $f' \cdot f' = f'$. Furthermore $f\mathcal{J}e = (x \cdot f \cdot y) \cdot e \cdot (x \cdot f \cdot y) \leq_\mathcal{J} f' \leq_\mathcal{J} f$. Hence $f\mathcal{J}f'$. It follows by Lemma 3.12 that $f' = f$. We now compute $e^\sharp = (x \cdot f \cdot f \cdot y)^\sharp = x \cdot f \cdot (f \cdot x \cdot y \cdot f)^\sharp \cdot f \cdot y = x \cdot f \cdot f^\sharp \cdot f \cdot y = x \cdot f^\sharp \cdot y$ (using consistency and $f = f'$).

This proves that $e\mathcal{J}f$ implies $e^\sharp \leq_\mathcal{J} f^\sharp$. Using symmetry, we obtain $e^\sharp\mathcal{J}f^\sharp$. □

Hence, if J is a regular $\mathcal{J}$-class, there exists a unique $\mathcal{J}$-class $J^\sharp$ which contains $e^\sharp$ for one/all idempotents $e \in J$. If $J = J^\sharp$, then J is called *stable*, otherwise, it is called *unstable*. The following lemma shows that stabilisation is trivial over stable $\mathcal{J}$-classes.

Lemma 3.14. *If J is a stable $\mathcal{J}$-class, then $e^\sharp = e$ for all idempotents $e \in J$.*

Proof. Indeed, we have $e^\sharp = e \cdot e^\sharp \cdot e$ and thus by Lemma 3.12, $e^\sharp = e$. □

The situation is different for unstable $\mathcal{J}$-classes. In this case, the stabilisation always goes down in the $\mathcal{J}$-order.

Lemma 3.15. *If J is an unstable $\mathcal{J}$-class, then $e^\sharp <_\mathcal{J} e$ for all idempotents $e \in J$.*

Proof. Since $e^\sharp = e \cdot e^\sharp$, it is always the case that $e^\sharp \leq_\mathcal{J} e$. Assuming J is unstable means that $e\mathcal{J}e^\sharp$ does not hold, which in turn implies $e^\sharp <_\mathcal{J} e$. □

We say that a word $u = a_1 \dots a_n$ in S^+ is *J-smooth*, for J a $\mathcal{J}$-class, if $u \in J^+$, and $\pi(u) \in J$. It is equivalent to say that $\pi(a_i a_{i+1} \cdots a_j) \in J$ for all $1 \leq i < j \leq n$. Remark that, according to Lemma 3.11, if J is irregular, J-smooth words have length at most 1. We will use the following lemma from [8] as a black-box. This is an instance of the factorisation forest theorem, but restricted to a single $\mathcal{J}$-class.

Lemma 3.16 (Lemma 14 in [8])**.** *Given a finite semigroup $\mathbf{S}$, one of its $\mathcal{J}$-classes J, and a J-smooth word u, there exists a Ramsey factorisation for u of height at most $3|J| - 1$.*

Remark that Ramsey factorisations and n-computations do only differ on what is allowed for a node of big degree, i.e., above n. That is why our construction makes use of Lemma 3.16 to produce Ramsey factorisations, and then based on the presence of big nodes, constructs a computation by gluing pieces of Ramsey factorisations.

Lemma 3.17. *Let J be a $\mathcal{J}$-class, u be a J-smooth word, and n be some non-negative integer. Then one of the two following items holds:*

(1) *there exists an n-computation for u of value $\pi(u)$ and height at most $3|J| - 1$, or;*
(2) *there exists an n-computation for some non-empty prefix w of u of value*[7] *$a <_\mathcal{J} J$ and height at most $3|J|$.*

Proof. Remark that if J is irregular, then u has length 1 by Lemma 3.11, and the result is straightforward. Remark also that if J is stable, and since the stabilisation is trivial in stable $\mathcal{J}$-classes (Lemma 3.14), every Ramsey factorisations for u of height at most $3|J| - 1$ (which exist by Lemma 3.16) are in fact n-computations for u.

[7]A closer inspection would reveal that $a \in J^\sharp$. This extra information is useless for our purpose.

The case of J unstable remains. Let us say that a node in a factorisation is *big* if its degree is more than n. Our goal is to correct the value of big nodes. If there is a Ramsey factorisation for u which has no big node, then it can be seen as an n-computation, and once more the first conclusion of the lemma holds.

Otherwise, consider the least non-empty prefix u' of u for which there is a Ramsey factorisation of height at most $3|J|-1$ which contains a big node. Let F be such a factorisation and x be a big node in F which is maximal for the descendant relation (there are no other big nodes below). Let F' be the subtree of F rooted in x. This decomposes u' into $vv'v''$ where v' is the factor of u' for which F' is a Ramsey factorisation. For this v', it is easy to transform F' into an n-computation T' for v': just replace the label e of the root of F' by $e^\sharp$. Indeed, since there are no other big nodes in F' than the root, the root is the only place which prevents F' to be an n-computation. Remark that from Lemma 3.15, the value of F' is $<_\mathcal{J} J$.

If v is empty, then v' is a prefix of u, and F' an n-computation for it. The second conclusion of the lemma holds.

Otherwise, by the minimality assumption and Lemma 3.16, there exists a Ramsey factorisation T for v of height at most $3|J|-1$ which contains no big node. Both T and T' being n-computations of height at most $3|J|-1$, it is easy to combine them into an n-computation of height at most $3|J|$ for vv'. This is an n-computation for vv', which inherits from F' the property that its value is $<_\mathcal{J} J$. It proves that the second conclusion of the lemma holds. □

We are now ready to establish Theorem 3.3.

Proof. The proof is by induction on the size of a left-right-ideal $Z \subseteq S$, i.e., $S \cdot Z \cdot S \subseteq S$ (remark that a left-right-ideal is a union of $\mathcal{J}$-classes). We establish by induction on the size of Z the following induction hypothesis:

IH: for all words $u \in Z^+ + Z^*S$ there exists an n-computation of height at most $3|Z|$ for u.

Of course, for $Z = S$, this proves Theorem 3.3.

The base case is when Z is empty, then u has length 1, and a single node tree establish the first conclusion of the induction hypothesis (recall that the convention is that the leaves do not count in the height, and as a consequence a single node tree has height 0).

Otherwise, assume Z non-empty. There exists a maximal $\mathcal{J}$-class J (maximal for $\leq_\mathcal{J}$) included in Z. From the maximality assumption, one can check that $Z' = Z \setminus J$ is again a left-right-ideal. Remark also that since Z is a left-right-ideal, it is downward closed for $\leq_\mathcal{J}$. This means in particular that every element a such that $a <_\mathcal{J} J$ belongs to Z'.

Claim: We claim ($\star$) that for all words $u \in Z^+ + Z^*S$,

(1) either there exists an n-computation of height $3|J|$ for u, or;
(2) there exists an n-computation of height at most $3|J|$ for some non-empty prefix of u of value in Z'.

Let w be the longest J-smooth prefix of u. If there exists no such non-empty prefix, this means that the first letter a of u does not belong to J. In this case, either $u = a$, and a is an n-computation witnessing the first conclusion of ($\star$). Otherwise a is not the last letter, and hence it belongs to Z. Since it does not belong to J, it belongs to Z'. In this case, a is an n-computation witnessing the second conclusion of ($\star$).

Otherwise, according to Lemma 3.17 applied to w, two situations can occur. The first case is when there is an n-computation T for w of value $\pi(w)$ and height at most $3|J|-1$.

There are several sub-cases. If $u = w$, of course, the n-computation T is a witness that the first conclusion of $(\star)$ holds. Otherwise, there is a letter a such that wa is a prefix of u. If $wa = u$, then $\pi(wa)[T, a]$ is an n-computation for wa of height at most $3|J|$, witnessing that the first conclusion of $(\star)$ holds. Otherwise, a has to belong to Z (because all letters of u have to belong to Z except possibly the last one). But, by maximality of w as a J-smooth prefix, either $a \in Z'$, or $\pi(wa) \in Z'$. Since Z' is a left-right-ideal, $a \in Z'$ implies $\pi(wa) \in Z'$. Then, $\pi(wa)[T, a]$ is an n-computation for wa of height at most $3|J|$ and value $\pi(wa) \in Z'$. This time, the second conclusion of $(\star)$ holds.

The second case according to Lemma 3.17 is when there exists a prefix v of w for which there is an n-computation of height at most $3|J|$ of value $<_{\mathcal{J}} J$. In this case, v is also a prefix of u, and the value of this computation is in Z'. Once more the second conclusion of $(\star)$ holds. This concludes the proof of Claim $(\star)$.

As long as the second conclusion of the claim $(\star)$ applied on the word u holds, this decomposes u into v_1u', and one can proceed with u'. In the end, we obtain that all words $u \in Z^+ + Z^*S$ can be decomposed into $u_1 \dots u_k$ such that there exist n-computations $T_1, \dots, T_k$ of height at most $3|Z|$ for $u_i, \dots, u_k$ respectively, and such that the values of $T_1, \dots, T_{k-1}$ all belong to Z' (but not necessarily the value of T_k). Let $a_1, \dots, a_k$ be the values of $T_1, \dots, T_k$ respectively. The word $a_1 \dots a_k$ belongs to $Z'^+ + Z'^*S$. Let us apply the induction hypothesis to the word $a_1 \dots a_k$. We obtain an n-computation T for $a_1 \dots a_k$ of height at most $3|Z'|$. By simply substituting $T_1, \dots, T_k$ to the leaves of T, we obtain an n-computation for u of height at most $3|J| + 3|Z'| = 3|Z|$. (Remark once more here that the convention is to not count the leaves in the height. Hence the height after a substitution is bounded by the sum of the heights.) □

3.5. **Comparing computations: the proof of Theorem 3.4.**

We now establish the second key theorem for computations, namely Theorem 3.4 which states that the result of computations is, in some sense unique. The proof works by a case analysis on the possible ways the over-computations and under-computations may overlap. We perform this proof for stabilisation monoids, thus using sm-computations. More precisely, all statements take as input computations, and output sm-computations, which can be then normalised into non-sm computations. The result for stabilisation semigroup can be derived from it. We fix ourselves from now on a stabilisation monoid $\mathbf{M}$.

Lemma 3.18. For all n-over-computations of value a over a word $u \in M^*$ of length at most n, $\pi(u) \leq a$.

Proof. By induction on the height of the over-computation, using the fact that an n-over-computation for a word of length at most n cannot contain a stabilisation node. □

Lemma 3.19. For all n-over-computations of value a over a word $a_1 \dots a_k$ ($k \geq 1$) such that $e \leq a_i$ for all i, and e is an idempotent, then $e^\sharp \leq a$.

Proof. By induction on the height of the over-computation. □

A sequence of $u_1, \dots, u_k$ is called a *decomposition* of u if $u = u_1 \dots u_k$. A decomposition is *strict* if all the u_i's are non-empty. We say that a non-leaf [under/over]-computation T for a word u *decomposes* u into $u_1, \dots, u_k$ if the subtree rooted at the first ith child of the root is an [under/over]-computation for u_i, for all i.

The core of the proof is contained in the following property:

Lemma 3.20. There exists a polynomial α such that for all $\alpha(n)$-over-computations T over a word u of value b, and all decompositions of u into $u_1,\dots,u_k$, then:

(1) there exist n-over-computations $B_1\dots B_k$ for $u_1,\dots,u_k$ respectively, of value $b_1,\dots,b_k$ respectively, and,
(2) there exists an n-over-computation B over $b_1\dots b_k$ of value b.

From this result, we immediately obtain a proof for Theorem 3.4.

Proof of Theorem 3.4. Let α be as in Lemma 3.20. Let α_p be the pth composition of α with itself. Let U be an n-under-computation of height at most p for some word u of value a, and T be an $\alpha_p(n)$-over-computation for u of value b. We aim at $a\leq b$. The proof is by induction on p.

If $p=0$, this means that u has length 1, then T and U are also restricted to a single leaf, and the result obviously holds. Otherwise, U decomposes u into $u_1,\dots,u_k$. let $x_1,\dots,x_k$ be the children of the root in U, and $a_1,\dots,a_k$ be their respective values. By applying Lemma 3.20 on T and the decomposition $u_1,\dots,u_k$. We construct the $\alpha_{p-1}(n)$-over-computations $B_1,\dots,B_k$ for $u_1,\dots,u_k$ respectively, and of respective values $b_1,\dots,b_k$, as well as an $\alpha_{p-1}(n)$-over-computation B of value b for $b_1,\dots,b_k$.

For all $i=1\dots k$, one can apply the induction hypothesis on U^i and B_i, and obtain that $a_i\leq b_i$. Depending on k, three cases have to be separated. If $k=2$ (binary node), then $a\leq a_1\cdot a_2\leq b_1\cdot b_2\leq b$. If $3\leq k\leq n$ (idempotent node), we have $a\leq a_1=\dots=a_k=e$ which is an idempotent. We have $e=a_i\leq b_i$ for all $i=1\dots k$. Hence by Lemma 3.18, $e\leq b$, which means $a\leq b$. If $k>n$ (stabilisation node), we have once more $a_1=\dots=a_k=e$ which is an idempotent, and such that $a\leq e^\sharp$. This time, by Lemma 3.19, we have $e^\sharp\leq b$. We obtain once more $a\leq b$. □

The remainder of this section is dedicated to the proof of Lemma 3.20. For shortening the proof, we will use the following terminology: one says that a word $u\in M^*$ *n-evaluates* to $a\in M$ if there exists an n-over-computation for u of value a. We will also say that $u_1,\dots,u_k$ n-evaluates to $b_1,\dots,b_k$ if u_i n-evaluates to b_i for all $i=1\dots k$. This notion is subject to elementary reasoning such that if $u_1,\dots,u_k$ n-evaluates to $b_1,\dots,b_k$ and $b_1\dots b_k$ n-evaluate to b, then $u_1\dots u_k$ n-evaluates to b (simply by plugging the over-computations together in the natural way).

Thus our goal is to prove that if $u_1\dots u_k$ $\alpha(n)$-evaluates to b then $u_1,\dots,u_k$ n-evaluates to $b_1,\dots,b_k$ where $b_1\dots b_k$ n-evaluates to b. This is equivalent to Lemma 3.20. We first establish this result for k fixed.

Lemma 3.21. For all positive integers k, there exists a polynomial α such that for all words u which $\alpha(n)$-evaluate to b, and all decompositions of u into $u_1\dots u_k$, then $u_1,\dots,u_k$ n-evaluates to $b_1,\dots,b_k$ such that $\pi(b_1\dots b_k)\leq b$.

Proof. Let us treat first the case $k=2$. Let $\alpha(n)=2n+1$ and let T be an $\alpha(n)$-over-computation of value b over a word $u=u_1u_2$. The proof is by induction on the height of T. Remark first that if u_1 or u_2 are empty, the result is straightforward. The case of a leaf means that either u_1 or u_2 are empty and hence this case is covered.

Otherwise the over-computation T decomposes u as $v_1,\dots,v_l$. Let $a_1,\dots,a_l$ be the values of the children of the root. There exist some s in $1\dots l$ and words w_1,w_2 such that:

$$u_1=v_1\dots v_{s-1}w_1\ ,\qquad v_s=w_1w_2,\qquad \text{and } u_2=w_2v_{s+1}\dots v_l\ .$$

One applies the induction hypothesis on v_s decomposed into w_1, w_2. From the induction hypothesis, we get that w_1, w_2 n-evaluate to c_1, c_2 such that $c_1 \cdot c_2 \le a_s$.

If $l = 2$ (binary node). Then either $s = 1$ or $s = 2$. The two cases being symmetric, let us treat the case $s = 1$. In this case, $u_1 = v_1, u_2 = v_2 w_2$ n-evaluate to $c_1, c_2 \cdot a_2$, and furthermore $\pi(c_1(c_2 \cdot a_2)) = (c_1 \cdot c_2) \cdot a_2 \le \pi(a_1 a_2)$.

If $3 \le l \le 2n+1$ (case of an idempotent node), then $a_1 = \dots = a_l = e$ idempotent, and $e \le b$. In this case $a_1 \dots a_{s-1}$ n-evaluates to e (or 1 if $s = 1$). We get from this that $v_1 \dots v_{s-1}$ n-evaluates to e (or 1 if $s = 1$) (since each u_i n-evaluates to a_i). Hence $u_1 = v_1 \dots v_{s-1} w_1$ n-evaluates to $b_1 = e \cdot c_1$ (or c_1 if $s = 1$). In the same way $u_2 = w_2 v_{s+1} \dots v_l$ n-evaluates to $b_2 = c_2 \cdot e$ (or c_2 if $s = l$). Since furthermore $c_1 \cdot c_2 \le e$, we obtain $b_1 \cdot b_2 \le e \le b$.

If $l > 2n+2$ (case of a stabilisation node), then $a_1 = \dots = a_l = e$ is an idempotent, and $e^\sharp \le b$. Two situations can occur depending on whether s is on the left or the right half of u. Let us treat the case $s > n+1$. In this case, $v_1 \dots v_{s-1}$ n-evaluates to $e^\sharp$, and thus $u_1 = v_1 \dots v_{s-1} w_1$ n-evaluates to $b_1 = e^\sharp \cdot c_1$. As in the idempotent case u_2 n-evaluates to $b_2 = c_2 \cdot e$ (or possibly c_2). Since $c_2 \cdot c_2 \le e$, we obtain $b_1 \cdot b_2 = e^\sharp \cdot c_1 \cdot c_2 \cdot e \le e^\sharp \le b$.

The case for $k > 2$ is easily obtained by induction on k from the above. Denote by α^p the pth composition of α with itself. Assume $u_1 \dots u_k$ $\alpha^p(n)$-evaluates to b, then $u_1 \dots u_{k-1}, u_k$ $\alpha^{p-1}(n)$ evaluate to c, b_k such that $c \cdot b_k \le b$ (using the case $k = 2$). Applying the induction hypothesis, we know that $u_1, \dots, u_{k-1}$ n-evaluate to $b_1, \dots, b_{k-1}$ such that $\pi(b_1 \dots b_{k-1}) \le c$. It follows that $u_1, \dots, u_k$ n-evaluate to $b_1, \dots, b_k$ with $\pi(b_1 \dots b_k) \le c \cdot b_k \le b$. □

Lemma 3.22. There exists K such that for all idempotents e, f, whenever $f \le a_i \cdot b_i$ for all $i = 1 \dots k$, $k \ge K$, and $b_i \cdot a_{i+1} \le e$, then $f^\sharp \le a_1 \cdot e^\sharp \cdot b_k$.

Proof. Thanks to the Theorem of Ramsey (since K is sufficiently large), there exists $i < j$ such that $x = \pi(b_i a_{i+1} \dots b_{j-1} a_j)$ is an idempotent. Clearly, $x \le e$, and hence $x^\sharp \le e^\sharp$. It follows that $f^\sharp \le a_i \cdot e^\sharp \cdot b_j$. Hence, since $f \le a_i \cdot b_i$, $f^\sharp \le \pi(a_1 b_1 \dots a_i) \cdot e^\sharp \cdot \pi(b_j \dots a_k)$. Since now $b_i \cdot a_{i+1} \le e$ for all $i = 1 \dots k-1$, we obtain $f^\sharp \le a_1 \cdot e^\sharp \cdot b_k$. □

Lemma 3.23. If T is a computation of value b for a word u, then $b \le \pi(u)$. In particular, for all words u and all n, u n-evaluates to $\pi(u)$.

Proof. By induction on T. □

Lemma 3.24. There exists a polynomial α such that, if $a_1, b_1, c_1, \dots, a_l, b_l, c_l$ are elements of M and e is an idempotent such that $a_i \cdot b_i \cdot c_i \le e$ for all $i = 1 \dots l$, then $b_1(c_1 \cdot a_2) b_2 \cdots b_{l-1}(c_{l-1} a_l) b_l$ n-evaluates to c such that:

- $a_1 \cdot c \cdot c_l \le e$,
- if $l > \alpha(n)$ or $(a_i \cdot b_i \cdot c_i) \le e^\sharp$ for some i then $a_1 \cdot c \cdot c_l \le e^\sharp$.

Proof. Remark first that by Lemma 3.23, $b_1(c_1 \cdot a_2) b_2 \cdots b_{l-1}(c_{l-1} a_l) b_l$ evaluates to $c = \pi(b_1 c_1 a_2 b_2 \dots b_l)$, and hence $a_1 \cdot c \cdot c_l = \pi(a_1 b_1 c_1 \dots c_l) \le e$. As a consequence, the first item holds. Furthermore, assume that $(a_i \cdot b_i \cdot c_i) \le e^\sharp$ for some i, then $a_1 \cdot c \cdot c_l = \pi(a_1 \dots c_l) \le e^\sharp$. In this case, the second conclusion holds.

So, from now, we are interested in the case $l > \alpha(n)$. Let K be the constant from Lemma 3.22. One chooses $\alpha(n) = (n + K + 1)^{3|M|}$.

Let us apply Theorem 3.3 for producing an $(n+K)$-computation T for the word $u = d_2 \dots d_{l-1}$ where $d_i = (b_i \cdot c_i \cdot a_{i+1})$ for all $i = 1 \dots l-1$, of value d and height at most $3|M|$. Remark that α has been chosen such that if $l > \alpha(n)$, then there is a stabilisation node of degree more than $n + K$ in T.

For this, let S be a subtree of T of value g, the root of which is a stabilisation node (hence of degree $l > n + K$). This subtree corresponds to the factor $v = d_i \dots d_{j-1}$. The computation S decomposes v into $v_1, \dots, v_l$. Let also f be the idempotent value shared by the children of the root in S. Each v_m is of the form $d_{k_m} \dots d_{k_{m+1}-1}$ where $i = k_1 < \dots < k_{l+1} = j$ Let $a'_i = \pi(b_{k_i} c_{k_i})$ and $b'_i = \pi(a_{k_i+1} b_{k_i+1} c_{k_i+1} \dots b_{k_{i+1}-1} c_{k_{i+1}-1} a_{k_{i+1}})$. By Lemma 3.23, $f \leq \pi(v_m) = a'_m \cdot b'_m$ for all $m = 1 \dots l$. Furthermore, $b'_m \cdot a'_{m+1} \leq e$ for all $m = 1 \dots l$. Hence one can apply Lemma 3.22, and get that $f^\sharp \leq a'_1 \cdot e^\sharp \cdot b'_l \leq b_{k_1} \cdot c_{k_1} \cdot e^\sharp \cdot a_{k_{l+1}-1}$. Hence v n-evaluates to $b_{k_1} \cdot c_{k_1} \cdot e^\sharp \cdot a_{k_{l+1}-1}$. It follows, again by Lemma 3.23 that u n-evaluates to $b_2 \cdot c_2 \cdot e^\sharp \cdot a_l$. It is easy to turn it into an n-computation for $b_1(c_1 \cdot a_2) b_2 \cdots b_{l-1}(c_{l-1} a_l) b_l$ since each $b_i(c_i \cdot a_{i+1})$ n-evalutates to d_i, and adding a new root (for b_l). The value c of this n-computation is such that $a_1 \cdot c \cdot c_l \leq e^\sharp$. □

We are now ready to conclude.

Proof of Lemma 3.20. One sets $\alpha(n)$ to be $n\alpha'(n)$ where α' is the polynomial provided by Lemma 3.24. Lemma 3.20 follows from the following induction hypothesis:

Induction hypothesis: For all words u which $\alpha(n)$-evaluate to b, and all decompositions of u into $u_1, \dots, u_k$ ($k \geq 2$), then $u_1, \dots, u_k$ n-evaluates to $b_1, \dots, b_k$, and $b_2 \dots b_{k-1}$ evaluates to c such that $b_1 \cdot c \cdot b_k \leq b$.

Induction parameter: The height of the $\alpha(n)$-over-computation T witnessing that $\alpha(n)$-evaluates to b.

The essential idea in the proof of the induction hypothesis is that T decomposes the word into $v_1, \dots, v_l$, and one has to study all the possible ways the v_i's and the u_j's may overlap. In practice, we will not talk about T anymore before the conclusion, but simply about a decomposition $v_1, \dots, v_l$. The intention must be clear that this $v_1, \dots, v_l$ will be the decomposition induced by T in the end.

From now on, let $v_1, \dots, v_l$ and $u_1, \dots, u_k$ be a decomposition of a word u. One assumes that each $v_1, \dots, v_l$ $\alpha(n)$-evaluates to $a_1, \dots, a_n$, and that one can apply the induction hypothesis for all $v_1, \dots, v_l$.

Binary nodes. If $l = 2$, then there exists s among $1 \dots k$ and words w, w' such that

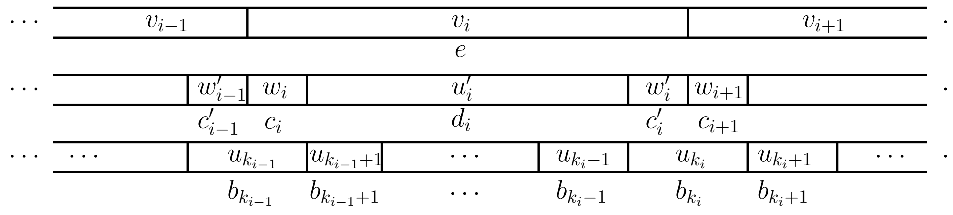
$$v_1 = u_1 \dots u_{s-1} w \ , \qquad u_s = ww' \ , \qquad \text{and} \quad v_2 = w' u_{s+1} \dots u_l \ .$$

One can apply the induction hypothesis to both v_1 and v_2. We obtain that $u_1, \dots, u_{s-1}, w$, $w', u_{s+1}, \dots, u_k$ n-evaluate to $b_1, \dots, b_{s-1}, d, d', b_{s+1}, \dots, b_k$, and that $b_2 \dots b_{s-1}, b_{s+1} \dots b_k$ n-evaluate to c_1, c_2 such that $b_1 \cdot c_1 \cdot d \leq a_1$ and $d' \cdot c_2 \cdot b_k \leq a_2$. It follows that u_s n-evaluates to $b_s = d \cdot d'$. Furthermore, $b_2 \dots b_{l-1}$ n-evaluates to $c_1 \cdot b_s \cdot c_2 = c$. Overall, $u_1, \dots, u_k$ n-evaluates to $b_1, \dots, b_k$ and $b_2 \dots b_{k-1}$ n-evaluates to c such that $b_1 \cdot c \cdot b_k = (b_1 \cdot c_1 \cdot d) \cdot (d' \cdot d_2 \cdot b_k) \leq a_1 \cdot a_2$.

Idempotent and stabilisation nodes. Assume now that $v_1, \dots, v_l$ $\alpha(n)$-evaluate to $e, \dots, e$, where e is idempotent. One aims at proving that $u_1, \dots, u_k$ n-evaluates to $b_1, \dots, b_k$, and $b_2 \dots b_{k-1}$ to c such that $b_1 \cdot c \cdot b_k \leq e$, and if l is sufficiently large (to be specified), $b_1 \cdot c \cdot b_k \leq e^\sharp$. We successively treat the two sub-cases, for which what is "sufficiently large" is different.

Idempotent and stabilisation nodes, first sub-case. Assume that none of the u_i's contain a v_j as a factor. In this case, "sufficiently large" means simply $\alpha'(n)$ (where α' is the polynomial from Lemma 3.24).

The assumption of this sub-case means that each v_i can be decomposed into $v_i = w_i u'_i w'_i$, where u'_i stands for $u_{k_{i-1}+1} \dots u_{k_i - 1}$, and such that $u_{k_i} = w'_i w_{i+1}$, where $1 = k_0 < k_2 < \dots < k_l = k$, $w_1 = u_1$ and $w'_l = u_k$.. This is illustrated in the following drawing:



Inside bars are the words, and just below their values obtained by their evaluation used during the proof.

One can apply the induction hypothesis for each v_i and $w_i, u_{k_{i-1}+1}, \dots, u_{k_i-1}, w'_i$. This proves that $w_i, u_{k_{i-1}+1}, \dots, u_{k_i-1}, w'_i$ n-evaluate to $c_i, b_{k_{i-1}+1}, \dots, b_{k_i-1}, c'_i$, and at the same time $b_{k_{i-1}+1} \dots b_{k_i-1}$ n-evaluates to d_i such that $c_i \cdot d_i \cdot c'_i \leq e$. Let us set now $b_{k_i} = c'_i \cdot c_{i+1}$. Clearly u_{k_i} n-evaluates to b_{k_i}. Let us also fix $b_1 = c_1$ and $b_k = c'_l$. We naturally get that $u_1, \dots, u_{k-1}$ n-evaluates to $b_1, \dots, b_k$.

Since $c_i \cdot d_i \cdot c'_i \leq e$ for all $i = 1 \dots l$, one can apply Lemma 3.24, proving that $d_1(c'_1 \cdot c_2)d_2 \dots d_{l-1}(c'_{l-1} \cdot c_l)d_l$ n-evaluates to some c such that $c_1 \cdot c \cdot c'_l \leq e$, and if furthermore l is sufficiently large, $c_1 \cdot c \cdot c'_l \leq e^\sharp$. This n-over computation is easily turned into an n-over computation over $b_2 \dots b_{k-1}$, by recalling that $b_{k_i} = c'_i \cdot c_{i+1}$, and that $b_{k_{i-1}+1} \dots b_{k_i-1}$ n-evaluates to d_i for all i. This terminates the study of the first case.

Idempotent and stabilisation nodes, second sub-case. Let us now treat the general case. This time, $u_1, \dots, u_k$ and $v_1, \dots, v_l$ may overlap in any manner. This time "sufficiently large" means above $\alpha(n)$.

We will consider intervals $I, J, \dots$ included in $\{1, \dots, l\}$. The width of an interval $[i, j]$ is simply $j - i + 1$. Given an interval $I = [i, j]$, one denotes by v_I the word $v_i \dots v_j$. Call a non-empty interval S *simple* if v_S is included in some u_m (by this, we refer to the interval of positions corresponding to the letters of v_S and u_m in the word u with respect to the decomposition $u = v_1 \dots v_l$ and $u = u_1 \dots u_k$ respectively). In this case, m is called the index of S. An interval is *nowhere simple* if it is nonempty and if none of its (non-empty) sub-intervals is simple. Remark that if $[i, j]$ and $[j+1, j']$ are simple then $[i, j']$ is simple. We can thus partition $[1, l]$ into intervals $S_0 < N_1 < \dots < N_m < S_m$ such that each S_h is simple, each N_h is nowhere simple, and $N_1, \dots, N_m$ are non-empty (only S_0 and S_m may be empty). Such a decomposition is obtained simply by considering the set of maximal simple intervals, and combining it with the decomposition of the remaining positions into maximal intervals. If necessary, one adds a first empty simple interval and/or a last simple empty interval.

Simple intervals. Each simple interval S_i has an index which is the index s_i such that v_{S_i} is a factor of u_{s_i}. Given i, define e_i to be e if $|S_i| \leq n$, and to be $e^\sharp$ otherwise. Remark that, by construction, v_{S_i} n-evaluates to e_i.

Nowhere simple intervals. We can decompose each v_{N_i} as $w_i, u_{s_{i-1}+1}, \dots, u_{s_i-1}, w'_i$ such that $u_{s_i} = w'_i v_{S_i} w_{i+1}$, with a particular case $u_0 = v_{S_0} w_1$, and $u_k = w'_m v_{S_m}$. For each of the nowhere simple intervals, one can apply the first sub-case. Hence, we know that $w_i, u_{s_{i-1}+1}, \dots, u_{s_i-1}, w'_i$ n-evaluate to some $c_i, b_{s_{i-1}+1}, \dots, b_{s_i-1}, c'_i$ and $b_{s_{i-1}+1} \dots b_{s_i-1}$ n-evaluates to d_i such that $c_i \cdot d_i \cdot c'_i \leq e$ $(\star)$.

Let us now set c''_i to be $c'_i \cdot e_i$. We apply Lemma 3.24, and deduce that the word

$$d_1(c''_1 \cdot c_2)d_2 \dots (c''_{m-2} \cdot c_{m-1})d_m$$

n-evaluates to some d such that $c_1 \cdot d \cdot c''_m \leq e$ (and sometimes $c_1 \cdot d \cdot c''_m \leq e^\sharp$). Remark that this n-over-computation can be transformed into a witness that $b_2, \dots, b_{k-1}$ n-evaluates to d, simply by recalling that u_{s_i} n-evaluates to $c'_i \cdot e_i \cdot c_{i+1} = c''_i \cdot c_{i+1}$, and that $b_{s_{i-1}+1} \dots b_{s_i-1}$ n-evaluates to d_i.

Thus what we need to prove is first that $b_1 \cdot d \cdot b_k \leq e$. But this is obvious since $b_1 \cdot d \cdot b_k \leq e_0 \cdot c_1 \cdot b_k \leq e \cdot e = e$. Assume now l is larger than $\alpha(n)$. Then, either $m > \alpha'(n)$, or there is some i such that $|S_i| > n$, i.e., $e_i = e^\sharp$. If this i is 0, then $b_1 \cdot d \cdot b_k \leq e_0 \cdot c_1 \cdot b_k \leq e^\sharp \cdot e = e^\sharp$. In all other situations, by Lemma 3.24, $c_1 \cdot d \cdot b_k \leq e^\sharp$, and as a consequence $b_1 \cdot d \cdot b_k \leq e_0 \cdot c_1 \cdot b_k \leq e^{\cdot} e^\sharp = e^\sharp$.

We can now conclude the proof of the induction hypothesis. Indeed, let T be an $\alpha(n)-$over-computation witnessing that u $\alpha(n)$-evaluates to some b. Then if T is restricted to a single leaf, the induction hypothesis obviously holds. Otherwise, T decomposes u into $v_1, \dots, v_l$, and each of $v_1, \dots, v_l$ is subject to the application of the induction hypothesis. Thus, one applies either the case of a binary node, or the general case of a stabilisation or idempotent node. In all cases, we obtain that $u_1, \dots, u_k$ n-evaluates to $b_1, \dots, b_k$ where $b_2 \dots b_{k-1}$ evaluates to c such that $b_1 \cdot c \cdot b_k \leq b$. This establishes the induction hypothesis. □

4. Recognisable cost functions

We have seen in the previous sections the notion of stabilisation monoids, as well as the key technical tools for dealing with them, namely computations, over-computations and under-computations. In particular, we have seen Theorem 3.3 and Theorem 3.4 which states the existence of computations and the "unicity" of their values. In this section, we put these notions in action, and introduce the definition of recognisable cost functions. We will see in particular that the hypothesis of Fact 2.8 is fulfilled by recognisable cost functions, and as a consequence the domination problem for cost-monadic logic is decidable over finite words.

4.1. Recognisable cost functions.

Let us fix a stabilisation monoid $\mathbf{M}$. An *ideal* of $\mathbf{M}$ is a subset I of M which is downward closed, *i.e.*, such that whenever $a \leq b$ and $b \in I$ we have $a \in I$. Given a subset $X \subseteq M$, we denote by $X\!\downarrow$ the least ideal which contains X, i.e., $X\!\downarrow = \{y \;:\; y \leq x,\; x \in X\}$.

The three ingredients used for recognising a cost functions are a monoid $\mathbf{M}$, a mapping h from letters of the alphabet $\mathbb{A}$ to M (that we extend into a morphism $\tilde{h}$ from $\mathbb{A}^*$ to M^*), and an ideal I.

We then define for each non-negative integer p four functions from $\mathbb{A}^*$ to $\mathbb{N} \cup \{\infty\}$:

$$\begin{aligned}
[\![\mathbf{M}, h, I]\!]^{--}_p(u) &= \inf\{n \;:\; \text{there exists an } n\text{-under-computation of value in } M \setminus I \text{ of height at most } p \text{ for } \tilde{h}(u)\;\}\\
[\![\mathbf{M}, h, I]\!]^{-}_p(u) &= \inf\{n \;:\; \text{there exists an } n\text{-computation of value in } M \setminus I \text{ of height at most } p \text{ for } \tilde{h}(u)\;\}\\
[\![\mathbf{M}, h, I]\!]^{+}_p(u) &= \sup\{n+1 \;:\; \text{there exists an } n\text{-computation of value in } I \text{ of height at most } p \text{ for } \tilde{h}(u)\;\}\\
[\![\mathbf{M}, h, I]\!]^{++}_p(u) &= \sup\{n+1 \;:\; \text{there exists an } n\text{-over-computation of value in } I \text{ of height at most } p \text{ for } \tilde{h}(u)\;\}
\end{aligned}$$

Those four functions (for each p) are candidates to be recognised by $\mathbf{M}, h, I$. The following lemma shows that if p is sufficiently large, all the four functions belong to the same cost function.

Lemma 4.1. For all $p \geq 3|M|$,

$$[\![\mathbf{M}, h, I]\!]_p^{--} \leq [\![\mathbf{M}, h, I]\!]_p^{-} \leq [\![\mathbf{M}, h, I]\!]_p^{+} \leq [\![\mathbf{M}, h, I]\!]_p^{++} \ .$$

For all p, there exists a polynomial α such that

$$[\![\mathbf{M}, h, I]\!]_p^{++} \leq \alpha \circ [\![\mathbf{M}, h, I]\!]_p^{--} \ .$$

For all $p \leq r$,

$$[\![\mathbf{M}, h, I]\!]_r^{--} \leq [\![\mathbf{M}, h, I]\!]_p^{--} \ , \qquad \text{and} \quad [\![\mathbf{M}, h, I]\!]_p^{++} \leq [\![\mathbf{M}, h, I]\!]_r^{++} \ .$$

Proof. All the inequalities are direct consequences of the results of the previous section.

The middle inequality in the first equation simply comes from the fact that, thanks to Theorem 3.3, the infimum in the definition of $[\![\mathbf{M}, h, I]\!]_p^{+}$ and the supremum in the definition of $[\![\mathbf{M}, h, I]\!]_p^{-}$ range over sets of integers which cover all integers. This suffices for justifying the inequality. The first and last inequalities of the equation simply come from the fact that each n-computation is in particular an n-under-computation and an n-over-computation respectively.

The second line is directly deduced from Theorem 3.4.

The third statement simply comes from the the fact that each n-under-computation of height at most p is also an n-under-computation of height r (left inequality). The same goes for over-computations. □

The main consequence of this lemma takes the form of a definition.

Definition 4.2. For all stabilisation monoids $\mathbf{M}$, all mappings from an alphabet $\mathbb{A}$ to M, and all ideals I, there exists a unique cost function $[\![\mathbf{M}, h, I]\!]$ from $\mathbb{A}^*$ to $\mathbb{N} \cup \{\infty\}$ such that for all $p \geq 3|M|$, all the functions $[\![\mathbf{M}, h, I]\!]_p^{--}$, $[\![\mathbf{M}, h, I]\!]_p^{-}$, $[\![\mathbf{M}, h, I]\!]_p^{+}$ and $[\![\mathbf{M}, h, I]\!]_p^{++}$ belong to $[\![\mathbf{M}, h, I]\!]$. It is called the *cost function recognised by* $\mathbf{M}, h, I$. One sometimes also writes that it is recognised by $\mathbf{M}$.

Example 4.1. The cost function $|\cdot|_a$ is recognised by $\mathbf{M}, \{a \mapsto a, b \mapsto b\}, \{0\}$, where $\mathbf{M}$ is the stabilisation monoid of Example 3.1. In particular, the informal reasoning developed in the example such as "a few+a few=a few" now has a formal meaning: the imprecision in such arguments is absorbed in the equivalence up to α of computation trees, and results in the fact that the monoid does not define a unique function, but instead directly a cost function.

Another example is the case of standard regular languages.

Example 4.2. A monoid $\mathbf{M}$ together with h from $\mathbb{A}$ to M and a subset $F \subseteq M$ *recognises* a language L over $\mathbb{A}$ if for all words u, $u \in L$ if and only if $\pi(\tilde{h}(u)) \in F$. The same monoid can be seen, thanks to Remark 3.3 as a stabilisation monoid. In this case, thanks to Remark 3.7, the same $\mathbf{M}, h, F$ recognises the characteristic mapping of L.

An elementary result is also the closure under composition with a morphism.

Fact 4.3. Let $\mathbf{M}, h, I$ recognise a cost function f over $\mathbb{A}^*$, and let z be a mapping from another alphabet $\mathbb{B}$ to $\mathbb{A}$, then $\mathbf{M}, h \circ z, I$ recognises $f \circ \tilde{z}$.

Proof. One easily checks that the computations involved in the definition of $[\![\mathbf{M}, h, I]\!]^{+}(\tilde{z}(u))$ are exactly the same as the one involved in the definition of $[\![\mathbf{M}, h \circ z, I]\!]^{+}(u)$. □

We continue this section by developing other tools for analysing the recognisable cost functions.

4.2. **The $\sharp$-expressions.** We now develop the notion of $\sharp$-expressions. This provides a convenient notation in several situations. This object was introduced by Hashiguchi for studying distance automata [17]. The $\sharp$-expressions can be seen in two different ways. On one side, a $\sharp$-expression allows to denote an element in a stabilisation monoid. On the other side, a $\sharp$-expression denotes an infinite sequence of words. Such sequences are used as witnesses, e.g., of the non-existence of a bound for a function (if the function tends toward infinity over this sequence), or of the non-divergence of a function f (if the function is bounded over the sequence). More generally, $\sharp$-expressions will be used as witnesses of non-domination.

In a finite monoid $\mathbf{M}$, given an element $a \in M$, one denotes by a^ω the only idempotent which is a power of a. This element does not exist in general for infinite monoids, while it always does for finite monoids (our case). Furthermore, when it exists, it is unique. In particular in a finite monoid $\mathbf{M}$, $a^\omega = a^{|M|!}$. This is a useful notion since the operator of stabilisation is only defined for idempotents. In a stabilisation monoid, let us denote by $a^{\omega\sharp}$ the element $(a^\omega)^\sharp$. As opposed to $a^\sharp$ which is not defined if a is not idempotent, $a^{\omega\sharp}$ is always defined.

A *$\sharp$-expression* over a set A is an expression composed of letters from A, products, and exponents with $\omega\sharp$. A $\sharp$-expression E over a stabilisation monoid denotes a computation in this stabilisation monoid. Its result is naturally an element of E, denoted *value*(E), and called the *value of* E. A $\sharp$-expression is called *strict* if it contains at least one occurrence of $\omega\sharp$.

Given a set $A \subseteq M$, call $\langle A\rangle^\sharp$ the set of values of expressions over A. Equivalently, it is the least set which contains A and is closed under product and stabilisation of idempotents. One also denotes $\langle A\rangle^{\sharp+}$ the set of values of strict $\sharp$-expressions over A.

The *n-unfolding* of a $\sharp$-expression over A is a word in A^+ defined inductively as follows:

$$\begin{aligned}
\text{unfold}(a,n) &= a\\
\text{unfold}(EF,n) &= \text{unfold}(E,n)\text{unfold}(F,n)\\
\text{unfold}(E^\sharp) &= \overbrace{\text{unfold}(E,n)\ldots\text{unfold}(E,n)}^{n \text{ times}}
\end{aligned}$$

We conclude this section by showing how $\sharp$-expressions can be used as witnesses of the behaviour of a regular cost function.

Proposition 4.3. Assume $\mathbf{M}, h, I$ recognises f, and let E be a $\sharp$-expression over $\mathbb{A}$ of value a, then:

- if $a \in I$ then $\{f(\text{unfold}(E,kn)) \;:\; n \geq 1\}$ tends toward infinity,
- if $a \notin I$ then $\{f(\text{unfold}(E,kn)) \;:\; n \geq 1\}$ is bounded.

where $k \geq 3$ is some multiple of $|M|!$.

Proof. We need, given a positive integer n, to produce an n-computation of value a. Assume $n \geq 3$, define the kn-computation by induction as follows:

$$\begin{aligned}
\text{computation}(a, n) &= a \\
\text{computation}(EF, n) &= \mathit{value}(EF)[\text{computation}(E, n), \text{computation}(F, n)] \\
\text{computation}(E^{\omega\sharp}) &= \mathit{value}(E^{\omega\sharp})[\overbrace{\text{computation}(E, n), \ldots, \text{computation}(E, n)}^{kn\ \text{times}}]
\end{aligned}$$

It is easy to check that computation(E, n) is a kn-compuation for unfold(u, kn), and that its value is *value*(E). In particular, requiring that k is a multiple of $|M|!$ ensures that $a^k = a^\omega$ is an idempotent and requiring $k \geq 3$ implies that stabilisation nodes indeed have degree three or more, as required by the definition. What we have is in fact stronger: computation(E, n) is an m-computation for unfold(E, n) for all $m \leq kn$. Furthermore computation(E, n) has its height bounded by $|E|$ (the size of the $\sharp$-expression).

Let us suppose $a \in I$. We have (where all heights are implicitly bounded by $|E|$):

$$\begin{aligned}
[\![\mathbf{M}, h, I]\!]^{+}_{|E|} &= \sup\{m \ :\ \text{there is an } m\text{-computation for } \tilde{h}(\text{unfold}(E, kn)) \text{ of value in } I\} \\
&\geq kn\ . \qquad\qquad (\text{using computation}(E, n) \text{ as a witness})
\end{aligned}$$

Thus $\{f(\text{unfold}(E, kn))\ :\ n \in \mathbb{N}\}$ tends toward infinity.

Similarly, if $a \notin I$, one obtains:

$$\begin{aligned}
[\![\mathbf{M}, h, I]\!]^{-}_{|E|} &= \inf\{m \ :\ \text{there is an } m\text{-computation for } \tilde{h}(\text{unfold}(E, kn)) \text{ of value in } M \setminus I\} \\
&= 0\ . \qquad\qquad (\text{using computation}(E, n) \text{ as a witness})
\end{aligned}$$

Thus $\{f(\text{unfold}(E, kn))\ :\ n \in \mathbb{N}\}$ is bounded. □

4.3. **Quotients, sub-stabilisation monoids and products.**

We introduce here some useful definitions for manipulating and composing stabilisation monoids. We use them for proving the closure of recognisable cost functions under min and max (Corollary 4.6).

Let $\mathbf{M} = \langle M, \cdot, \sharp, \leq\rangle$ and $\mathbf{M}' = \langle M', \cdot', \sharp', \leq'\rangle$ be stabilisation monoids. A *morphism of stabilisation monoids* from $\mathbf{M}$ to $\mathbf{M}'$ is a mapping μ from M to M' such that

- $\mu(1_{\mathbf{M}}) = 1_{\mathbf{M}'}$,
- $\mu(x) \cdot' \mu(y) = \mu(x \cdot y)$ for all x, y in M,
- for all x, y in M, if $x \leq y$ then $\mu(x) \leq' \mu(y)$,
- $\mu(e)^{\sharp'} = \mu(e^\sharp)$ for all $e \in E(\mathbf{M})$, (in this case, $\mu(e) \in E(\mathbf{M}')$).

Remark 4.4. The n-computations (*resp.*, n-under-computations, n-over-computations) over $\mathbf{M}$ are transformed by morphism (applied to each node of the tree) into n-computations (*resp.*, n-under-computations, n-over-computations) over $\mathbf{M}'$. In a similar way, the image under morphism of a $\sharp$-expression over $\mathbf{M}$ is a $\sharp$-expression over $\mathbf{M}'$.

We immediately obtain:

Lemma 4.4. For μ a morphism of stabilisation monoids from $\mathbf{M}$ to $\mathbf{M}'$, h a mapping from an alphabet $\mathbb{A}$ to $\mathbf{M}$ and I' an ideal of $\mathbf{M}'$, we have:

$$[\![\mathbf{M}, h, \mu^{-1}(I')]\!] = [\![\mathbf{M}', \mu \circ h, I']\!]\ .$$

Proof. Let us remark first that $I = \mu^{-1}(I')$ is an ideal of $\mathbf{M}$.

Let u be a word in $\mathbb{A}^*$. Let us consider an n-computation over $\mathbf{M}$ for $\tilde{h}(u)$ of value $a \in \mu^{-1}(I')$. This computation can be transformed by morphism into an n-computation over $\mathbf{M}'$ for $(\mu \circ h)(u)$ of value $\mu(a) \in I'$. In a similar way, each n-computation over $\tilde{h}(u)$ of value $a \in M' \setminus \mu^{-1}(I')$ can be transformed into an n-computation of value $\mu(a) \in M' \setminus I'$. □

The notion of morphism is intimately related to the notion of product. Given two stabilisation monoids $\mathbf{M} = \langle M, \cdot, \sharp, \leq \rangle$ and $\mathbf{M}' = \langle M', \cdot', \sharp', \leq' \rangle$, one defines their *product* by:

$$\mathbf{M} \times \mathbf{M}' = \langle M \times M', \cdot'', \sharp'', \leq'' \rangle$$

where $(x, x') \cdot'' (y, y') = (x \cdot y, x' \cdot' y')$, $(e, e')^{\sharp''} = (e^\sharp, e'^{\sharp'})$, and $(x, x') \leq'' (y, y')$ if and only if $x \leq y$ and $x' \leq' y'$.

As expected, the projection over the first component (*resp.*, second component) is a morphism of stabilisation monoids from $\mathbf{M} \times \mathbf{M}'$ onto $\mathbf{M}$ (*resp.*, onto $\mathbf{M}'$). It follows by Lemma 4.4 that if f is recognised by $\mathbf{M}, h, I$ and g by $\mathbf{M}', h', I'$, then f is also recognised by $\mathbf{M} \times \mathbf{M}', h \times h', I \times M'$ and g by $\mathbf{M} \times \mathbf{M}', h \times h', M \times I'$, in which one sets $(h \times h')(a) = (h(a), h(a'))$ for all letters a. Thus one obtains:

Lemma 4.5. If f and g are recognisable cost functions over $\mathbb{A}^*$, there exists a stabilisation monoid $\mathbf{M}$, an application h from $\mathbb{A}$ to $\mathbf{M}$ and two ideals I, J such that $\mathbf{M}, h, I$ recognises f and $\mathbf{M}, h, J$ recognises g.

Corollary 4.6. If f and g are recognisable, then so are $\max(f, g)$ and $\min(f, g)$.

Proof. According to Lemma 4.5, one assumes f recognised by $\mathbf{M}, h, I$ and g by $\mathbf{M}, h, J$. Then (for a height fixed to at most $p = 3|M|$) one has:

$$\begin{aligned}
&\max([\![\mathbf{M}, h, I]\!]^+_p, [\![\mathbf{M}, h, J]\!]^+_p)(u) \\
&\quad = \max \begin{cases} \sup\{n+1 \ : \ \text{there is an } n\text{-computation for } \tilde{h}(u) \text{ of value in } I\} \\ \sup\{n+1 \ : \ \text{there is an } n\text{-computation for } \tilde{h}(u) \text{ of value in } J\} \end{cases} \\
&\quad = \sup\{n+1 \ : \ \text{there exists an } n\text{-computation over } \tilde{h}(u) \text{ of value in } I \cup J\} \\
&\quad = [\![\mathbf{M}, h, I \cup J]\!]^+_p \ .
\end{aligned}$$

Thus $\max(f, g)$ is recognised by $\mathbf{M}, h, I \cup J$. In a similar way, $\min(f, g)$ is recognised by $\mathbf{M}, h, I \cap J$. □

4.4. **Decidability of the domination relation.** We are now ready to establish the decidability of the domination relation.

Theorem 4.7. The domination relation ($\preccurlyeq$) is decidable over the regular cost functions.

Proof. Let f, g be recognisable cost functions. According to Lemma 4.5 there exists a stabilisation monoid $\mathbf{M}$, a mapping h from the alphabet $\mathbb{A}$ to M and two ideals I, J such that $\mathbf{M}, h, I$ recognises f and $\mathbf{M}, h, J$ recognises g.

We show that f dominates g if and only if the following (decidable) property holds:

$$\langle h(\mathbb{A}) \rangle^\sharp \cap I \subseteq J \ .$$

First direction. Let us suppose $\langle h(\mathbb{A})\rangle^\sharp \cap I \subseteq J$. Of course, every n-computation over $\tilde{h}(u)$ for a word u has its value in $\langle h(\mathbb{A})\rangle^\sharp$. It follows that (for heights at most $3|M|$):

$$\begin{aligned}[\![\mathbf{M}, h, I]\!]^+(u) &= \sup\{n+1 \;:\; \text{there is an } n\text{-computation for } h(u) \text{ of value in } I\}\\ &\le \sup\{n+1 \;:\; \text{there is an } n\text{-computation for } h(u) \text{ of value in } J\}\\ &= [\![\mathbf{M}, h, J]\!]^+(u)\ .\end{aligned}$$

Thus we have $f \preccurlyeq g$.

Second direction. Let us suppose the existence of $a \in \langle h(\mathbb{A})\rangle^\sharp \cap I \setminus J$. By definition of $\langle h(\mathbb{A})\rangle^\sharp$, there is a $\sharp$-expression E over $h(\mathbb{A})$ of value a. Let F be the $\sharp$-expression over $\mathbb{A}$ obtained by substituting to each element $x \in h(\mathbb{A})$ some letter from $c \in \mathbb{A}$ such that $h(c) = x$. According to Proposition 4.3, f is unbounded over $\{\text{unfold}(F, kn) \;:\; n \ge 1\}$ (for some suitable k). However, still applying Proposition 4.3, g is bounded over $\{\text{unfold}(F, kn) \;:\; n \ge 3\}$. This witnesses that g does not dominate f. □

4.5. Closure under inf-projection.

We establish the following theorem.

Theorem 4.8. Recognisable cost functions are effectively closed under inf-projection.

The projection in the classical case (of regular languages) requires a powerset construction (for deterministic automata or for monoids). In our case, the approach is similar. Let h be a mapping from alphabet $\mathbb{A}$ to $\mathbb{B}$. The goal of a stabilisation monoid which would recognise the inf-projection by z of a recognisable cost function is to keep track of all the values a computation could have taken for some inverse image by $\tilde{z}$ of the input word. Hence an element of the stabilisation monoid for the inf-projection of the cost function consists naturally of a set of elements in the original monoid.

A closer inspection reveals that it is possible to close those subsets downward, i.e., consider only ideals. In fact, it is not only possible, but it is even necessary for the construction to go through. Let us describe formally this construction.

We have to consider a construction of ideals. Let $M_\downarrow$ be the set of ideals of $\mathbf{M}$. One equips $M_\downarrow$ of an order simply by inclusion:

$$I \le J \qquad \text{if} \qquad I \subseteq J\ ,$$

and of a product as follows:

$$A \cdot B \quad = \quad \{a \cdot b \;:\; a \in A,\ b \in B\}{\downarrow}\ .$$

Finally, the stabilisation is defined for an idempotent by:

$$E^\sharp = \langle E\rangle^{\sharp+}{\downarrow}\ .$$

The resulting structure $\langle M_\downarrow, \cdot, \le, \sharp\rangle$ is denoted $\mathbf{M}_\downarrow$.

It may seem a priori that our first goal would be to prove that the structure defined in the above way is indeed a stabilisation monoid. In fact, thanks to Proposition 3.5, this will be for free (see Lemma 4.12 below).

We now prove that $\mathbf{M}_\downarrow$ can be used for recognising the inf-projection of a cost function recognised by $\mathbf{M}$. Thus our goal is to relate the (under)-computations in $\mathbf{M}_\downarrow$ to the (under)-computations in $\mathbf{M}$. This will provide a semantic link between the two stabilisation monoids. This relationship takes the form of Lemmas 4.10 and 4.11 below.

Let us first state a simple remark on the structure of idempotents.

Lemma 4.9. If E is an idempotent in $M_{\downarrow}$, then all $a \in E$ are such that $a \leq b \cdot e \cdot c$ where $b, c, e \in E$ and e is an idempotent.

Proof. As $E = E \cdots E$, for all n, there exist $a_1, \dots, a_n \in E$ such that $a \leq a_1 \cdots a_n$. Using Ramsey's theorem, for n sufficiently large, there exist $1 < i \leq j < n$ such that $a_i \cdots a_j = e$ is an idempotent. One sets $b = a_1 \cdots a_{i-1}$, $c = a_{j+1} \cdots a_n$. We have $b, c, e \in E$, and $a \leq b \cdot e \cdot c$. □

Lemma 4.10. Let $A_1 \dots A_k$ be a word over $M_{\downarrow}$ and let T be an n-under-computation over $A_1 \dots A_k$ of height at most p and of value A. For all $a \in A$, there exists an n-under-computation of height $3p$ and value a for some word $a_1 \dots a_k$ such that $a_1 \in A_1$,…,$a_k \in A_k$.

Proof. The proof is by induction on p.

Leaf case, i.e., $T = A_1$. Let $a \in A \subseteq A_1$, then a is an n-computation of value $a \in A$.

Binary node, i.e., $T = A[T_1, T_2]$. Let B_1 and B_2 be the respective values of T_1 and T_2. Let $a \in A \subseteq B_1 \cdot B_2$. By definition of the product, there exists $b_1 \in B_1$ and $b_2 \in B_2$ such that $a \leq b_1 \cdot b_2$. By the induction hypothesis, there exist n-computations t_1 and t_2 of respective values b_1 and b_2. The n-under-computation $a[t_1, t_2]$ satisfies the induction hypothesis.

Idempotent node. $T = F[T_1, \dots, T_k]$ for some $k \leq n$ where $F \subseteq E$ for an idempotent E such that the value of T_i is E for all i. Let $a \in F \subseteq E$. We have $a \leq b \cdot e \cdot c$ for some $b, c, e \in E$ (Lemma 4.9). We then apply the induction hypothesis for $b, e, \dots, e$ and c on the computations $T_1, \dots, T_{k-1}$ and T_k respectively, yielding the n-under-computations $t_1, \dots, t_{k-1}$ and t_k respectively. Constructing the n-under-computation $a[t_1, (e \cdot c)[e[t_2, \dots, t_{k-1}], t_k]]$ concludes.

Stabilisation node. $T = F[T_1, \dots, T_k]$ for some $k > n$ and $F \subseteq E^{\sharp}$ for some idempotent E such that the value of T_i is E for all i. Let $a \in F \subseteq E^{\sharp}$. We have $a \leq b \cdot e \cdot c$ for some $b, e, c \in E$ (Lemma 4.9). We then apply the induction hypothesis for $b, e, \dots, e$ and c respectively and the computations $T_1, \dots, T_k$ respectively, yielding the n-under-computations $t_1, \dots, t_k$ respectively. We conclude by constructing the n-under-computation $a[t_1, (e^{\sharp} \cdot c)[e^{\sharp}[t_2, \dots, t_{k-1}], t_k]]$ (remark that $e^{\sharp}[t_2, \dots, t_{k-1}]$ is a valid under-computations since $e^{\sharp} \leq e$). □

Lemma 4.11. Let $A_1 \dots A_k$ be a word over $M_{\downarrow}$ and T be a $n^{3|M|}$-over-computation for $A_1 \dots A_k$ of height p and value A. For all $a_1 \in A_1, \dots, a_k \in A_k$, there exists an n-computation over $a_1 \dots a_k$ of value $a \in A$ and of height at most $3|M|p$.

Proof. The proof is by induction on p.

Leaf case, i.e., $T = A$ *and* $u = a_1 \in A_1 \subseteq A_1$. Hence a_1 is a computation satisfying the induction hypothesis.

Binary node, i.e., $T = A[T_1, T_2]$ where T_1 and T_2 have respective values B_1 and B_2 such that $B_1 \cdot B_2 \subseteq A$. One applies the induction hypothesis on T_1 and T_2, and gets computations t_1 and t_2, of respective values $b_1 \in B_1$ and $b_2 \in B_2$. The induction hypothesis is then fulfilled with the n-computation $(b_1 \cdot b_2)[t_1, t_2]$ of value $b_1 \cdot b_2 \in A$.

Idempotent node, i.e., $T = F[T_1, \dots, T_k]$ for $k \leq n^{3|M|}$ where $T_1, \dots, T_k$ share the same idempotent value $E \subseteq F$. Let $t_1, \dots, t_k$ be the n-computations of respective values $b_1, \dots, b_k$ obtained by applying the induction hypothesis on $T_1, \dots, T_k$ respectively. Furthermore, according to Theorem 3.3, there exists an n-computation t for the word $b_1 \dots b_k$ of height at most $3|M|$. Let a be the value of t. Since E is an idempotent, it is closed under product and stabilisation and contains $b_1, \dots, b_k$. It follows that $a \in E$ (by induction on the height of t). We conclude using the n-computation $t\{t_1, \dots, t_k\}$ (obtained from t by substituting the ith leaf of t for t_i) which satisfies the induction hypothesis.

Stabilisation node, i.e., $T = F[T_1, \dots, T_k]$ for $k > n^{3|M|}$ where $T_1, \dots, T_k$ all share the same idempotent value $E \subseteq F$. Let $t_1, \dots, t_k$ be the n-computations of respective values $b_1, \dots, b_k$ obtained by applying the induction hypothesis on $T_1, \dots, T_k$ respectively. Furthermore, according to Theorem 3.3, there exists an n-computation t for $b_1 \dots b_k$ of height at most $3|M|$. Since t has height at most $3|M|$ and has more than $n^{3|M|}$ leaves, it contains at least one node of degree more than n, namely, a stabilisation node. It is then easy to prove by induction on the height of t that the value of t belongs to $\langle E\rangle^{\sharp+}$. Thus the n-computation $t\{t_1, \dots, t_k\}$ satisfies the induction hypothesis. □

Lemma 4.12. $\mathbf{M}_\downarrow$ is a stabilisation monoid.

Proof. Consider an n-under-computation T of value A in $\mathbf{M}_\downarrow$ for some word $A_1 \dots A_k$ of height at most p, and some $\alpha(n)$-over-computation T' for the same word of value B (with $\alpha(n) = \alpha'(n)^{3|M|}$ where α' is obtained from Theorem 3.4 applied to $\mathbf{M}$ for height at most $3p$). We aim at $A \le B$. Indeed, this implies that one can use Proposition 3.5, and get that $\mathbf{M}_\downarrow$ is a stabilisation semigroup (it is then straightforward to prove it a stabilisation monoid.

Let $a \in A$, we aim at $a \in B$, thus proving $A \subseteq B$, i.e., $A \le B$. By Lemma 4.10, there exists an n-under-computation for some word $a_1 \dots a_k$ with $a_1 \in A_1, \dots, a_k \in A_k$, of height at most $3p$ and value a. Applying Lemma 4.11 on T', there exists an $\alpha'(n)$-over-computation for the same word $a_1 \dots a_k$ of value $b \in B$. Then applying Theorem 3.4, we get $a \le b$. Hence $a \in B$ since B is downward closed. □

We can now establish the result of this section.

Proof of Theorem 4.8. Assume a function f over the alphabet $\mathbb{A}$ is recognised by $\mathbf{M}, h, I$ and let z be some mapping from $\mathbb{A}$ to $\mathbb{B}$. One aims at proving that $f_{\inf,z}$ (recall that it is the function which maps each word u over $\mathbb{B}$ to $\inf\{f(v) : \tilde{z}(v) = u\}$) is recognised by $\mathbf{M}_\downarrow$. For this, let H from $\mathbb{B}$ to $M_\downarrow$ map $b \in \mathbb{B}$ to $h(z^{-1}(b))\!\downarrow$, and let $K \subseteq M_\downarrow$ be

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

Let F be defined for all word u over $\mathbb{B}$.

$$\begin{aligned}
&[\![\mathbf{M}_\downarrow, H, K]\!]^{--}_{3|M_\downarrow|} \\
&= \inf\{n \ : \ \text{there is an } n\text{-under-computation for } \tilde{H}(u) \text{ of value } A \not\subseteq I \text{ of height } 3|M_\downarrow|\} \\
&\ge \inf\{n \ : \ \text{there is an } n\text{-under-computation for } v \in \tilde{H}(u) \text{ of value } a \in M \setminus I \text{ of height } 9|M_\downarrow|\} \\
&\hspace{25em}\text{(according to Lemma 4.10)} \\
&= ([\![\mathbf{M}, h, I]\!]^{--}_{9|M_\downarrow|})_{\inf,z} \ .
\end{aligned}$$

Conversely,

$$\begin{aligned}
&\left\lfloor \sqrt[3|M_\downarrow|]{[\![\mathbf{M}_\downarrow, H, K]\!]^{++}_{3|M_\downarrow|}} \right\rfloor\\
&= \sup\{n+1 \ :\ \text{there is an } n^{3|M|}\text{-over-computation for } \tilde{H}(u) \text{ of value } A \subseteq I \text{ of height } 3|M_\downarrow|\}\\
&\leq \sup\{n+1 \ :\ \text{there is an } n\text{-over-computation for } v \in \tilde{H}(u) \text{ of value } a \in M \setminus I\\
&\qquad\qquad\qquad \text{of height } 9|M||M_\downarrow|\} \qquad\qquad \text{(according to Lemma 4.11)}\\
&= ([\![\mathbf{M}, h, I]\!]^{++}_{9|M||M_\downarrow|})_{\inf,z}\ .
\end{aligned}$$

□

4.6. **Closure under** sup**-projection.**

We now establish the closure under sup-projection, using a proof very similar to the previous section.

Theorem 4.13. Recognisable cost functions are effectively closed under sup-projection.

The closure under sup-projection follows the same principle as the closure under inf-projection. It uses also a power set construction. However, since everything is reversed, this is a construction of co-ideals. A *co-ideal* is a subset of a stabilisation monoid which is upward closed. Let $\uparrow(\mathbf{M})$ be set of co-ideals over a given stabilisation monoid $\mathbf{M}$. Given a set A, let us denote by $A\!\uparrow$ the least co-ideal containing A, i.e., $A\!\uparrow = \{y \ :\ y \geq x \in A\}$. One equips $\uparrow(\mathbf{M})$ of an order by:

$$I \leq J \qquad \text{if} \quad I \supseteq J\ ,$$

of a product with:

$$A \cdot B \quad = \quad \{a \cdot b \ :\ a \in A,\ b \in B\}\!\uparrow\ ,$$

and of a stabilisation operation by:

$$E^\sharp = \langle E \rangle^{\sharp+}\!\uparrow\ .$$

Let us call $\mathbf{M}_\uparrow$ the resulting structure.

The proof is extremely close to the case of inf-projection. However, a careful inspection would show that all computations of bounds, and even some local arguments need to be modified. However, we have preferred to duplicate it for clarity.

Let us state a simple remark on the structure of idempotents.

Lemma 4.14. If E is an idempotent in $M_\uparrow$, then all $a \in E$ are such that $a \geq b \cdot e \cdot c$ where $b, c, e \in E$ and e is an idempotent.

Proof. As $E = E \cdots E$, for all n, there exist $a_1, \dots, a_n \in E$ such that $a \geq a_1 \cdots a_n$. Using Ramsey's theorem, for n sufficiently large, there exist $1 < i \leq j < n$ such that $a_i \cdots a_j = e$ is an idempotent. One sets $b = a_1 \cdots a_{i-1}$, $c = a_{j+1} \cdots a_n$. We have $b, c, e \in E$, and $a \geq b \cdot e \cdot c$. □

Lemma 4.15. Let $A_1 \dots A_k$ be a word over $\mathbf{M}_\uparrow$ and let T be an $(n+2)$-over-computation over $A_1 \dots A_k$ of height at most p and of value A. For all $a \in A$, there exists an n-over computation of height $3p$ and value a for some word $a_1 \dots a_k$ such that $a_1 \in A_1, \dots, a_k \in A_k$.

Proof. The proof is by induction on p.

Leaf case, i.e., $T = A_1$. Let $a \in A \subseteq A_1$, then a is an n-computation of value $a \in A$.

Binary node, i.e., $T = A[T_1, T_2]$. Let B_1 and B_2 be the respective values of T_1 and T_2. Let $a \in A \subseteq B_1 \cdot B_2$. By definition of the product, there exists $b_1 \in B_1$ and $b_2 \in B_2$ such that $a \geq a_1 \cdot a_2$. By the induction hypothesis, there exist n-computations t_1 and t_2 of respective values b_1 and b_2. The n-over-computation $a[t_1, t_2]$ satisfies the induction hypothesis.

Idempotent node. $T = F[T_1, \dots, T_k]$ for some $k \leq n+2$ where $F \subseteq E$ for an idempotent E such that the value of T_i is E for all i. Let $a \in F \subseteq E$. We have $a \geq b \cdot e \cdot c$ for some $b, c, e \in E$ (Lemma 4.14). We then apply the induction hypothesis for $b, e, \dots, e$ and c an the computations $T_1, \dots, T_{k-1}$ and T_k respectively, yielding the n-under-computations $t_1, \dots, t_{k-1}$ and t_k respectively. We conclude by constructing the n-under-computation $a[t_1, (e \cdot c)[e[t_2, \dots, t_{k-1}], t_k]]$.

Stabilisation node. $T = F[T_1, \dots, T_k]$ for some $k > n+2$ and $F \subseteq E^\sharp$ for some idempotent E such that the value of T_i is E for all i. Let $a \in F \subseteq E^\sharp$. We have $a \geq b \cdot e \cdot c$ for some $b, c, e \in E$ (Lemma 4.14). We then apply the induction hypothesis for $b, e, \dots, e$ and c respectively an the computations $T_1, \dots, T_k$ respectively, yielding the n-over-computations $t_1, \dots, t_k$ respectively. We conclude by constructing the n-over-computation $a[t_1, (e^\sharp \cdot c)[e^\sharp[t_2, \dots, t_{k-1}], t_k]]$ (remark in particular that the interval $\{2 \dots k-1\}$ has length at least $n+1$). □

Lemma 4.16. Let $A_1 \dots A_k$ be a word over $\mathbf{M}_\uparrow$ and T be an n-under-computation T for $A_1 \dots A_k$ of height p and value A. For all words $u = a_1 \dots a_k$ with $a_1 \in A_1, \dots, a_k \in A_k$, there exists an n-computation over $a_1 \dots a_k$ of value $a \in A$ and of height at most $3|M|p$.

Proof. The proof is by induction on p.

Leaf case, i.e., $T = A$ and $u = a_1 \in A_1 \subseteq A$. Hence a_1 is a computation satisfying the induction hypothesis.

Binary node, i.e., $T = A[T_1, T_2]$ where T_1 and T_2 have respective values B_1 and B_2 such that $B_1 \cdot B_2 \subseteq A$. One applies the induction hypothesis on T_1 and T_2, and get computations t_1 and t_2, of respective values $b_1 \in B_1$ and $b_2 \in B_2$. The induction hypothesis is then fulfilled with the n-computation $(b_1 \cdot b_2)[t_1, t_2]$ of value $b_1 \cdot b_2 \in A$.

Idempotent node, i.e., $T = F[T_1, \dots, T_k]$ for $k \leq n$ where $T_1, \dots, T_k$ share the same idempotent value $E \subseteq F$. Let $t_1, \dots, t_k$ be the n-computations of respective values $b_1, \dots, b_k$ obtained by applying the induction hypothesis on $T_1, \dots, T_k$ respectively. Furthermore, according to Theorem 3.3, there exists an n-computation t for the word $b_1 \dots b_k$ of height at most $3|M|$. Let a be the value of t. Since E is an idempotent, it is closed under product and contains $b_1, \dots, b_k$. Since furthermore t does not contain any node of stabilisation, we obtain that $a \in E$ (by induction on the height of t). We conclude using the n-computation $t\{t_1, \dots, t_k\}$ (obtained from t by substituting the ith leaf of t for t_i) which satisfies the induction hypothesis.

Stabilisation node, i.e., $T = F[T_1, \dots, T_k]$ for $k > n$ where $T_1, \dots, T_k$ all share the same idempotent value $E \subseteq F$. Let $t_1, \dots, t_k$ be the n-computations of respective values $b_1, \dots, b_k$

obtained by applying the induction hypothesis on $T_1, \dots, T_k$ respectively. Furthermore, according to Theorem 3.3, there exists an n-computation t for $b_1 \dots b_k$ of height at most $3|M|$ and value a. Since $E^\sharp$ is a sub-stabilisation monoid of $\mathbf{M}$ which contains $b_1, \dots, b_k$, a also belongs to $E^\sharp$. Thus the n-computation $t\{t_1, \dots, t_k\}$ satisfies the induction hypothesis. □

Lemma 4.17. $\mathbf{M}_\uparrow$ is a stabilisation monoid.

Proof. Consider an n-under-computation T of value A in $\mathbf{M}_\uparrow$ for some word $A_1 \dots A_k$ of height at most p, and some $\alpha(n)$-over-computation T' for the same word of value B (with $\alpha(n) = \alpha'(n) + 2$ where α' is obtained from Theorem 3.4 applied to $\mathbf{M}$ for height at most $3p$). We aim at $A \subseteq B$. Indeed, this implies that one can use Proposition 3.5, and get that $\mathbf{M}_\uparrow$ is a stabilisation monoid.

Let $b \in B$, we aim at $b \in A$, thus proving $B \subseteq A$, i.e., $A \leq B$. By Lemma 4.15, there exists an $\alpha'(n)$-over-computation for some word $a_1 \dots a_k$ with $a_1 \in A_1, \dots, a_k \in A_k$, of height at most $3p$ and value b. Applying Lemma 4.16 on T, there exists an n-computation for the same word $a_1 \dots a_k$ of value $a \in A$. Then applying Theorem 3.4, we get $a \leq b$. Hence $b \in B$ since B is upward closed. □

We are ready to complete the proof of closure under sup-projection.

Proof. Proof of Theorem 4.13 Assume a function f over the alphabet $\mathbb{A}$ is recognised by $\mathbf{M}, h, I$ and let z be some mapping from $\mathbb{A}$ to $\mathbb{B}$. One aims at proving that $f_{\sup,z}$ is recognised by $\mathbf{M}_\uparrow$. For this, let H from $\mathbb{B}$ to $M_\uparrow$ map $b \in \mathbb{B}$ to $h(z^{-1}(b))\uparrow$, and let $K \subseteq M_\uparrow$ be

$$K = \{J \in M_\uparrow \ : \ J \cap I = \emptyset\} \ .$$

Let F be defined for all words u over $\mathbb{B}$.

$$\begin{aligned}
&[\![\mathbf{M}_\uparrow, H, K]\!]^{++}_{3|M_\uparrow|}\\
&= \sup\{n \ : \ \text{there is an } n\text{-over-comp. for } \tilde{H}(u) \text{ of value } A \not\subseteq I \text{ of height } 3|M_\uparrow|\}\\
&\leq \sup\{n \ : \ \text{there is an } n\text{-over-comp. for } v \in \tilde{H}(u) \text{ of value } a \in M \setminus I \text{ of height } 9|M_\uparrow|\}\\
&\hfill \text{(according to Lemma 4.15)}\\
&= ([\![\mathbf{M}, h, I]\!]^{++}_{9|M_\uparrow|})_{\sup,z}
\end{aligned}$$

The converse inequality is obtained as follows

$$\begin{aligned}
&[\![\mathbf{M}_\uparrow, H, K]\!]^{--}_{3|M_\uparrow|}\\
&= \inf\{n \ : \ \text{there is an } n\text{-under-comp. for } \tilde{H}(u) \text{ of value } A \subseteq I \text{ of height } 3|M_\uparrow|\}\\
&\geq \inf\{n \ : \ \text{there is an } n\text{-under-comp. for } v \in \tilde{H}(u) \text{ of value } a \in I \text{ of height } 9|M||M_\uparrow|\}\\
&\hfill \text{(according to Lemma 4.16)}\\
&= ([\![\mathbf{M}, h, I]\!]^{--}_{3|M||M_\uparrow|})_{\sup,z}
\end{aligned}$$

□

References

[1] Parosh Aziz Abdulla, Pavel Krcál, and Wang Yi. R-automata. In *CONCUR*, pages 67–81. Springer, 2008.

[2] Sebastian Bala. Regular language matching and other decidable cases of the satisfiability problem for constraints between regular open terms. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 596–607. Springer, 2004.

[3] Achim Blumensath, Martin Otto, and Mark Weyer. Boundedness of monadic second-order formulae over finite words. In *36th ICALP*, Lecture Notes in Computer Science, pages 67–78. Springer, July 2009.

[4] Mikolaj Bojanczyk. Weak mso with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011.

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

[6] Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *36th ICALP*, number 5556 in Lecture Notes in Computer Science, pages 139–150. Springer, 2009.

[7] Thomas Colcombet. Factorisation forests for infinite words and applications to countable scattered linear orderings. *Theoretical Computer Science*, 411:751–764, 2010.

[8] Thomas Colcombet. Green's relations and their use in automata theory. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2011.

[9] Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In *CSL*, number 5213 in Lecture Notes in Computer Science, pages 416–430. Springer, 2008.

[10] Thomas Colcombet and Christof Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In *35th ICALP*, number 5126 in Lecture Notes in Computer Science, pages 398–409. Springer, 2008.

[11] Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.

[12] Françoise Dejean and Marcel-Paul Schützenberger. On a question of Eggan. *Information and Control*, 9(1):23–25, 1966.

[13] Lawrence C. Eggan. Transition graphs and the star-height of regular events. *Michigan Math. J.*, 10:385–397, 1963.

[14] Gösta Grahne and Alex Thomo. Approximate reasoning in semistructured data. In *KRDB*, 2001.

[15] Pierre A. Grillet. *Semigroups. An introduction to the structure theory.* Pure and Applied Mathematics, Marcel Dekker. 193. New York, NY: Marcel Dekker, Inc. ix, 398 p, 1995.

[16] Kosaburo Hashiguchi. A decision procedure for the order of regular events. *Theoretical Computer Science*, 8:69–72, 1979.

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

[18] Kosaburo Hashiguchi. Regular languages of star height one. *Information and Control*, 53(3):199–210, 1982.

[19] Kosaburo Hashiguchi. Representation theorems on regular languages. *J. Comput. Syst. Sci.*, 27(1):101–115, 1983.

[20] 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.

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

[22] Kosaburo Hashiguchi. Algorithms for determining relative inclusion star height and inclusion star height. *Theor. Comput. Sci.*, 91(1):85–100, 1991.

[23] Kosaburo Hashiguchi. New upper bounds to the limitedness of distance automata. *Theor. Comput. Sci.*, 233(1–2):19–32, 2000.

[24] Karel Culik II and Jarkko Kari. Image compression using weighted finite automata. In *MFCS*, pages 392–402, 1993.

[25] Daniel Kirsten. Desert automata and the finite substitution problem. In *STACS*, volume 2996 of *Lecture Notes in Computer Science*, pages 305–316. Springer, 2004.

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

[27] Daniel Kirsten. A burnside approach to the finite substitution problem. *Theoretical Computer Science*, 39(1):15–50, 2006.

[28] Daniel Kirsten. Distance desert automata and star height substitutions. Habilitation, Universität Leipzig, Fakultät für Mathematik und Informatik, 2006.

[29] Daniel Kirsten. On the complexity of the relative inclusion star height problem. *Advances in Computer Science and Engineering*, 5(2):173–211, 2010.

[30] Manfred Kufleitner. The height of factorization forests. In *MFCS*, volume 5162, pages 443–454, 2008.

[31] Gérard Lallement. *Semigroups and Combinatorial Applications*. Wiley, 1979.

[32] Hing Leung. *An Algebraic Method for Solving Decision Problems in Finite Automata Theory*. PhD thesis, Pennsylvania State University, Department of Computer Science, 1987.

[33] Hing Leung. On the topological structure of a finitely generated semigroup of matrices. *Semigroup Forum*, 37:273–287, 1988.

[34] Hing Leung. Limitedness theorem on finite automata with distance functions: An algebraic proof. *Theoretical Computer Science*, 81(1):137–145, 1991.

[35] Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, 2004.

[36] Robert McNaughton. The loop complexity of pure-group events. *Information and Control*, 11(1-2):167–176, 1967.

[37] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[38] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[39] Jean-Eric Pin. *Varieties of Formal Languages*. North Oxford Academic, London and Plenum, New York, 1986.

[40] Imre Simon. Limited subsets of a free monoid. In *FOCS*, pages 143–150. IEEE, 1978.

[41] Imre Simon. Recognizable sets with multiplicities in the tropical semiring. In *MFCS*, volume 324 of *Lecture Notes in Computer Science*, pages 107–120. Springer, 1988.

[42] Imre Simon. Factorization forests of finite height. *Theoretical Computer Science*, 72:65–94, 1990.

[43] Imre Simon. On semigroups of matrices over the tropical semiring. *RAIRO ITA*, 28(3-4):277–294, 1994.

[44] Imre Simon. A short proof of the factorization forest theorem. *Tree Automata and Languages*, pages 433–438, 92.

[45] Szymon Toruńczyk. *Languages of profinite words and the limitedness problem*. PhD thesis, Warsaw University, 2011.

[46] Andreas Weber. Distance automata having large finite distance or finite ambiguity. *Mathematical Systems Theory*, 26(2):169–185, 1993.

[47] Andreas Weber. Finite-valued distance automata. *Theoretical Computer Science*, 134(1):225–251, 1994.