

# THE DIMENSIONALITY OF QUOTIENT ALGEBRAS

CHRIS KROOK

Once we have found a Gröbner basis  $\mathcal{GB}$  for an ideal  $\mathcal{I}$  in some free algebra  $\mathcal{U}$ , we can consider the quotient algebra  $Q = \mathcal{U}/\langle \mathcal{GB} \rangle$ . It is not guaranteed that this quotient algebra will always be finite dimensional. In fact, there are many cases where the quotient algebra is infinite dimensional and has polynomial or even exponential growth. We present some functions that investigate the dimensionality of the quotient algebra.

Before continuing we observe that instead of looking at  $Q$  we can also look at the monomial algebra  $Q' := \mathcal{U}/\langle \mathcal{GB}' \rangle$  where  $\mathcal{GB}' := \{ Lt(b) \mid b \in \mathcal{GB} \}$ . We call  $Q'$  the associated algebra for  $Q$ . Although in general  $Q'$  will be a different algebra than  $Q$ , they both have the same basis, since  $Q$  and  $Q'$  give rise to the same *normal words*. The dimensionality of the quotient algebra and other concepts of interest such as the Hilbert series are equal for both the algebras  $Q$  and  $Q'$  since they depend only on the bases. For this reason we will restrict ourselves to the study of monomial algebras in the remainder of this paper.

## 1. THE FINCHECK-ALGORITHM: DETERMINE WHETHER THE QA IS FINITE OR INFINITE DIMENSIONAL.

These monomial algebras will be determined by an alphabet  $\mathcal{A}$  and a set of monomials  $\mathcal{M}$ , which is reduced such that none of the monomials in  $\mathcal{M}$  divides another monomial in  $\mathcal{M}$ . Both the alphabet and the set of monomials are assumed to be finite. Furthermore, all  $m \in \mathcal{M}$  are words of finite length in  $\mathcal{A}^*$ , which is the space of all words over the alphabet  $\mathcal{A}$  of finite or infinite length.

**Definition 1.** *We call a monomial  $u$  normal (w.r.t. some reduced set  $\mathcal{M}$ ) if  $\mathcal{M} \nmid u$ . By this notation we mean that no monomial in  $\mathcal{M}$  divides  $u$ . We also speak of normal words.*

The basis of the monomial algebra will consist of all normal words w.r.t.  $\mathcal{M}$ . this basis is infinite if and only if it is possible to construct a normal word  $w \in \mathcal{A}^*$  of infinite length. It is obvious that this condition is sufficient, since the word  $w$  will contain infinitely many different normal subwords. The other direction follows directly from the assumption that our alphabet is finite. Thus up to a given length, only finitely many words exist. This provides us with a direct and intuitive approach to determine finiteness given  $\mathcal{A}$  and  $\mathcal{M}$ . We simply try to construct an infinite normal word. If we succeed, we can conclude infiniteness of the monomial algebra. Otherwise we conclude finiteness. Let's make this more precise.

We will introduce the notion of a graph of normal words and use this graph to obtain some information on the structure of infinite words.

**Definition 2.** Given an alphabet  $\mathcal{A}$  and a set of monomials  $\mathcal{M}$ , we can define the Ufnarowski Graph<sup>1</sup>,  $G_U$ . Its vertex set  $V$  consists of all normal words  $w \in \mathcal{A}^{l_{\mathcal{M}}-1}$ , where  $l_{\mathcal{M}} := -1 + \max_{m \in \mathcal{M}} |m|$ . For each  $v, w \in V$  there is a directed edge  $v \rightarrow w$  if and only if there exist  $a, b \in \mathcal{A}$  s.t.  $va = bw$  and  $\mathcal{M} \nmid va$ .

As one readily checks there is a one-to-one correspondence between paths of length  $l$  in  $G_U$  and normal words of length  $l + l_{\mathcal{M}}$ . This implies that each infinite normal word corresponds to an infinite path in  $G_U$ . Since the graph has a finite number of vertices, due to the finiteness of  $\mathcal{A}$  and  $\mathcal{M}$ , this implies that such an infinite path must contain a cycle. But then the word corresponding to merely repeating this cycle is an infinite normal word as well.

**Remark 1.1.** If there exists an infinite word that is normal w.r.t.  $\mathcal{M}$ , then either it is cyclic or it gives rise to a cyclic infinite word that is also normal w.r.t.  $\mathcal{M}$ .

We will use this in the proof of the following lemma.

**Lemma 1.2.** If there exists an infinite word  $w' \in \mathcal{A}^*$  that is normal w.r.t.  $\mathcal{M}$ , then there also exists a cyclic infinite word  $w \in \mathcal{A}^*$  that is normal w.r.t.  $\mathcal{M}$  s.t.

$$(1) \quad \forall_{r,s \geq 1} w[1 \dots s] \leq w[r \dots r+s-1].$$

Before giving the proof we introduce the notation  $u \triangleleft v$  to denote that  $u$  is a prefix of  $v$  and the notation  $u \trianglelefteq v$  to denote that  $u$  is a proper prefix of  $v$ . Furthermore  $u^t$  denotes taking  $t$  concatenations of a word  $u$ .

**Proof:** Let  $w' \in \mathcal{A}^*$  be infinite and normal w.r.t.  $\mathcal{M}$ . Then according to remark 1.1,  $w'$  gives rise to a cyclic infinite word  $w'' = v'^{\infty}$ , where  $v' \in \mathcal{A}^p$  for some finite  $p > 0$ . We assume that  $v$  is the lexicographically smallest cyclic shift of  $v'$ . Then there is a  $u \triangleleft v'$  s.t.  $v'^{\infty} = uv^{\infty}$ . Now define  $w := v^{\infty}$  and the lemma follows immediately.  $\square$

**Remark 1.3.** For a given word  $u$  that is lexicographically smaller than all its cyclic shifts, the word  $w = u^q u'$  with  $u' \trianglelefteq u$  and  $q \geq 1$ , satisfies condition (1). This is an immediate consequence of the property of  $u$ .

**Corollary 1.4.** In order to find infinite words, it is sufficient only to use words satisfying (1). This implies that in its construction, given a word  $w \in \mathcal{A}^l$  satisfying (1) we have the following two ways to proceed;

- if  $\mathcal{M} \nmid w$  then  $w := w'$  where  $w'$  is the smallest word satisfying  $w' > w$  and  $|w'| \leq |w|$ . It is obvious that  $w'$  will also satisfy condition (1) since the symbol increased will be the last position of  $w'$ ;

**Example:**  $\mathcal{A} = \{x, y\}, xyxy \in \mathcal{M}$

$\dots \rightarrow xyxy \rightarrow xyx \rightarrow \dots$

- if  $\mathcal{M} \nmid w$  then we want to extend the potential beginning of an infinite word, such that (1) remains satisfied. For this purpose we can use a pointer that moves along the word while lengthening it and is reset to the first position if the word is increased. The pointer points to the next symbol to be added. By doing so,  $w$  will always have the form mentioned in remark 1.3, thus

---

<sup>1</sup>This graph should not be confused with the graph of normal words  $G_N$  which is generally used for another type of graph. See for  $G_U$  f.e. [1]

satisfying condition (1) . In the following example we denote the pointer position using bold letters.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xxx\}$   
 $\dots \rightarrow \mathbf{x}xy \rightarrow x\mathbf{x}yx \rightarrow xy\mathbf{y}xx \rightarrow xy\mathbf{y}xy \rightarrow \dots$

This leads to the following algorithm for constructing infinite words. Denote the alphabet by  $\mathcal{A} = \{a_1, \dots, a_k\}$ .

**Algorithm 1.5. - construct an infinite word**

- (1) Build a tree  $\mathcal{T}$  of reversed monomials from  $\mathcal{M}$ .
- (2) Start with the smallest word  $w := a_1$ . Set a pointer  $p$  at the first position, thus  $p := 1$ .
- (3) Check whether  $\mathcal{M}|w$  ( $\star$ ):
  - if  $w$  is normal, then we:
    - (a) check if we can conclude infiniteness ( $\boxtimes$ ) in which case we **terminate** instantly;
    - (b) extend word:  $w := w + w[p]$  and  $p := p + 1$ .
  - if  $w$  is not normal, then we:
    - (a) check if  $w[1] = a_k$  in which case we can conclude finiteness and **terminate** instantly;
    - (b) increase  $w$  in a minimal way. i.e.  $w := u$  where  $u$  is the smallest word satisfying  $w < u$  and  $|u| \leq |w|$ . Furthermore the pointer is reset to  $p := 1$ .
- (4) Repeat step 3 until **termination** is achieved.

**ad  $\star$ :** While constructing our word  $w$  we already know that the longest proper prefix  $w'$  of  $w$  is normal, thus we only have to check whether each suffix  $v$  of  $w$  is normal. By use of a tree structure  $\mathcal{T}$  to store the reversed monomials from  $\mathcal{M}$  this comes down to checking whether there is no branch  $b$  in  $\mathcal{T}$  s.t.  $b \triangleleft \text{rev}(v)$ . This will always take at most  $\text{length}(v)$  comparisons. (*end of  $\star$* )

**ad  $\boxtimes$ :** We still need to work out how to conclude infiniteness from a word  $w$  such that  $\mathcal{M} \nmid w$ . For this purpose we remember the Ufnarowski graph from definition 2 and in particular the one-to-one correspondence between infinite paths in  $G_U$  (i.e. cycles) and infinite words. This tells us how to adapt our algorithm to detect infiniteness. While constructing our word  $w$ , we simultaneously build up that part of  $G_U$  that is on our route. Now we can easily check our word for cycles, by just checking whether the vertex we want to add to our graph is already in the vertex set. Let us give an example.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xx, yyy\}$

$w = \mathbf{x}$	$V = \{\}$	
$w = x\mathbf{x}$	$V = \{\}$	$\mathcal{M} w$
$w = \mathbf{x}y$	$V = \{xy\}$	
$w = xy\mathbf{x}$	$V = \{xy, yx\}$	
$w = xy\mathbf{y}$	$V = \{xy, yx\}$	<b>DONE</b>

In the final step, we find a cycle and conclude infiniteness. The bold letters indicate the position of the pointer.

Algorithm 1.5 actually automatically keeps track of the route in  $G_U$  since all the words on this route can simply be read from the word  $w$  itself. (*end of ✖*)

**Lemma 1.6.** *Algorithm 1.5 terminates and concludes whether the monomial algebra determined by an alphabet  $\mathcal{A}$  and a set of monomials  $\mathcal{M}$ , ordered lexicographically, is finite or infinite.*

**Proof:** We need to show correct termination of the algorithm. In each step the word  $w$  is increased lexicographically. Since  $G_U$  consists of at most  $k^{l_{\mathcal{M}}}$  vertices, a word length greater than  $k^{l_{\mathcal{M}}} + l_{\mathcal{M}}$ , which corresponds to a path of length greater than  $k^{l_{\mathcal{M}}}$  in  $G_U$  implies a cycle and thus infiniteness. Thus assuming that an infinite word exists, this will be found after a finite number of steps.

Now assume that all words are finite. There are only finitely many words with length  $\leq k^{l_{\mathcal{M}}} + l_{\mathcal{M}}$  thus in a finite number of steps  $w[1] = a_k$  will hold. Notice that from this point on, increasing  $w = (a_k)^q$  gives us  $w = (a_k)^{q+1}$  and after finitely many steps  $\mathcal{M}|w$  will hold which implies finiteness since all attempts to create infinite words have failed.  $\square$

We will consider two small examples, showing how the algorithm works in both the finite and the infinite case. Denoted are all words that are considered by the algorithm.

**Example: (infinite)**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xx, xyx, yyy\}$

$w = \mathbf{x}$	$V = \{\}$	
$w = x\mathbf{x}$	$V = \{\}$	$\mathcal{M} w$
$w = \mathbf{xy}$	$V = \{xy\}$	
$w = xy\mathbf{x}$	$V = \{xy\}$	$\mathcal{M} w$
$w = xy\mathbf{y}$	$V = \{xy, yy\}$	
$w = xy\mathbf{yx}$	$V = \{xy, yy, yx\}$	
$w = xy\mathbf{yxy}$	$V = \{xy, yy, yx\}$	<i>DONE</i>

We conclude infiniteness since we encounter a cycle. The infinite normal word is  $w = (xyy)^\infty$ . Notice that increasing the word  $xx$  in step 3, ensures that from that point on no words containing  $xx$  will be considered. Therefore in the final step the word  $xyyxy$  is checked in instead of first checking  $xyyxx$ . One can easily think of examples where this principle reduces the number of words to be considered much more.

**Example: (finite)**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xx, xyx, yyy\}$

$w = \mathbf{x}$	$V = \{\}$	
$w = x\mathbf{x}$	$V = \{\}$	$\mathcal{M} w$
$w = \mathbf{xy}$	$V = \{xy\}$	
$w = xy\mathbf{x}$	$V = \{xy, yx\}$	
$w = xy\mathbf{xy}$	$V = \{xy, yx\}$	$\mathcal{M} w$
$w = xy\mathbf{y}$	$V = \{xy, yy\}$	
$w = xy\mathbf{yx}$	$V = \{xy, yy, yx\}$	
$w = xy\mathbf{yxy}$	$V = \{xy, yy, yx\}$	$\mathcal{M} w$
$w = xy\mathbf{yyy}$	$V = \{xy, yy\}$	$\mathcal{M} w$
$w = \mathbf{y}$	$V = \{\}$	
$w = y\mathbf{y}$	$V = \{yy\}$	
$w = yy\mathbf{y}$	$V = \{yy\}$	$\mathcal{M} w, \text{DONE}$

We can conclude finiteness. Notice that in the case of finiteness we will always have to consider all the possible words satisfying condition (1), having no proper prefix dividable by  $\mathcal{M}$ .

We have seen that in the finite case, our algorithm has to check many words. One might wonder if there are also some bad scenarios in the infinite case. Unfortunately there are. We will describe them using the Ufnarovski Graph  $G_U$ .

**Lemma 1.7.** *Let  $|\mathcal{A}| = k$ . Then for each  $l \geq 1$  there exists a set  $\mathcal{M}$  such that  $l_{\mathcal{M}} = l$  and  $G_U$  contains only a cycle of length  $k^l$ , thus visiting all the vertices in  $G_U$ .*

**Proof:** The cycle we're looking for is well known and well studied and called the *De Bruijn-cycle*. Construct a graph  $G$  which has as vertex set all possible words of length  $l - 1$ . There is a directed edge from vertex  $u$  to vertex  $v$  if and only if there exist  $a, b \in \mathcal{A}$  such that  $ua = bv$ . Label this edge with  $b$ . Now notice that for each vertex  $v$  we have  $\deg_{in}(v) = \deg_{out}(v) = k$ . Therefore there must exist an Euler cycle in  $G$  and one readily sees that writing down the labels in this cycle gives us exactly the *De Bruijn-cycle* we're looking for, which is indeed a cycle of length  $k \cdot k^{l-1}$ .

Now given the cycle in the graph  $G_U$ , it's easy to define the set  $\mathcal{M}$  s.t.  $\mathcal{M}$  induces  $G_U$ , just by preventing all edges not on the cycle to exist.  $\square$

As an unwelcome result of lemma 1.7, there exists always a bad case in which the length of the shortest cycle and thus the amount of comparisons needed to conclude infiniteness grows exponentially in  $l_{\mathcal{M}}$ , roughly said the length of the largest monomial in  $\mathcal{M}$ .

## 2. THE EPFINCHECK-ALGORITHM: DETERMINE THE GROWTH OF THE QA

Sometimes we want to know more about the dimensionality of our algebra than just whether it has a finite or infinite basis. Then we are interested in the actual growth of the algebra. We will give a definition of the concept growth and we will present an algorithm to compute it.

**Definition 3.** *Given a monotone function  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  we define the growth of  $f$  as the equivalence class  $[f]$  of  $f$  where*

$$[f] = \{g : \mathbb{N} \rightarrow \mathbb{R}^+ \mid \exists_{c_1, c_2, m_1, m_2 \in \mathbb{N}} \forall_{n \in \mathbb{N}} \frac{1}{c_1} g\left(\frac{n}{m_1}\right) \leq f(n) \leq c_2 g(m_2 n)\}$$

Furthermore we can distinguish the following cases:

- if  $[f(n)] \leq [1]$  then we say that  $f$  has finite growth;
- if  $[f(n)] \leq [n^d]$  for some  $d > 0$  and  $d$  is a minimal natural number satisfying this inequality, then we call  $f$  polynomial of degree  $d$ ;
- if  $[f(n)] \geq [a^n]$  for  $a > 1$ , then we call  $f$  exponential.

### Growth of an algebra:

We need to relate this definition of growth with algebras, so we need to define a useful function on our algebra. In section 1 we were interested whether a monomial algebra induced by an alphabet  $\mathcal{A}$  and a set of monomials  $\mathcal{M}$  was finite or not, i.e. whether the number of normal words of finite length was finite. This hints us in the direction of a nice growth function on algebras. We will let our function count

the number of words of up to each given length  $n$ . More formally, we consider our algebra  $A$  which is graded, i.e.  $A = \bigoplus_i A_i$ , where  $A_i$  is the subspace consisting of all words of length  $i$ . Now define  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  by  $f(n) = \dim(\sum_{i=1}^n A_i)$ .

### Growth of a graph:

In practice however, we will translate the problem of computing the growth of the algebra, into the problem of computing the growth of the graph  $G_U$ . So we will also define a growth function on graphs. Let  $G$  be a finite graph. The intended function counts the number of different paths in  $G$  up to each given length  $n$ . Thus we define  $f : \mathbb{N} \rightarrow \mathbb{R}^+$  by  $f(n) = \#\text{different paths in } G \text{ of length up to } n$ .

Note that our algebra corresponds to the graph  $G_U$  in such a way that there exists a bijection between words of length  $n$  in the algebra and paths of length  $n$  in the graph. Thus these notions of growth are equivalent.

One readily sees that if a graph contains a cycle, there exist paths of length  $n$  for all  $n \in \mathbb{N}$ , thus the graph must be at least polynomial. We can intuitively see a relation between the growth of a graph and the number of cycles in it.

- If a finite graph contains no cycles, then the total number of different paths is finite and thus the graph has finite growth.
- If a finite graph contains a path visiting  $d$  non intersecting cycles, then it has at least polynomial growth of degree  $d$ . Intuitively, there are  $k + d - 1$  road segments,  $k$  of which are cycles and  $d - 1$  of which are connecting paths between the cycles. The number of paths that contain  $k$  cycles equals the number of ways to appoint  $k$  cycles in a set of  $k + d - 1$  road segments which equals  $\text{Binomial}(k + d - 1, k)$  which is polynomial in  $k$ . Since the length of the cycles is upper bounded, this gives us polynomial growth in the length of the paths as well.
- If a graph contains 2 intersecting cycles, then the number of different paths of length up to  $n$  grows exponentially in  $n$ . Intuitively each time you reach a vertex  $v$  that is contained in both cycles, you can go 2 ways. So there are already  $2^k$  different paths that visit  $v$   $k$  times.

This topic is dealt with more thoroughly in [2]. There one can also find a proof of the above intuitive remark.

**2.1. Constructing an infinite word.** We will next present an algorithm that is based on algorithm 1.5 and that can distinguish the three cases; finiteness, polynomial growth and exponential growth. But in case of polynomial growth it doesn't always compute the exact degree, but will upper and lower bound this.

#### **Algorithm 2.1. - construct an infinite word and look for intersecting cycles**

- (1) Build a tree  $\mathcal{T}$  of reversed obstructions from  $\mathcal{M}$ .
- (2) Start with the smallest word  $w := a_1$ . Set a pointer  $p$  at the first position, thus  $p := 1$ . Furthermore let  $\text{cycles} := \{\}$ .
- (3) Check whether  $\mathcal{M}|w$  ( $\star$ ):
  - if  $w$  is normal, then we:
    - (a) if  $w$  contains a cycle of subwords of length  $l_{\mathcal{M}}$ :

- that intersects with a cycle in the list *cycles* then we can conclude exponential growth and **terminate** immediately.
- that doesn't intersect with any cycle in the list *cycles*, then we add a pair of indices  $\{i_1, i_2\}$  to the list *cycles* s.t.  $w[i_1 \dots i_2 + l_{\mathcal{M}}]$  is the cycle.
- (b) extend word:  $w := w + w[p]$  and  $p := p + 1$ .
- if  $w$  is not normal, then we:
  - (a) check if  $w[1] = a_k$  in which case we have checked all words. We can conclude polynomial growth of degree  $d_{low} \leq d \leq d_{upp}$  (**✕**) and will **terminate** immediately.  $d = 0$  corresponds to finiteness.
  - (b) – Increase  $w$  in a minimal way. i.e.  $w := u$  where  $u$  is the smallest word satisfying  $w < u$  and  $|u| \leq |w|$ .
    - The pointer is reset to  $p := 1$ .
    - Furthermore the list *cycles* is updated as to only consider beginnings of cycles in the new word  $u$ . Thus for each pair  $\{i_1, i_2\}$  in *cycles*, it is deleted if  $|u| \leq i_1$  or it is replaced by  $\{i_1, |u| - l_{\mathcal{M}} + 1\}$  if  $i_1 < |u| \leq i_2 + l_{\mathcal{M}}$ .
- (4) Repeat step 3 until **termination** is achieved.

**ad ★:** See the remark at algorithm 1.5 for more details on how  $\mathcal{T}$  is used for this.  
(end of ★)

**ad ✕:** As we mentioned earlier our algorithm only gives an upper bound and a lower bound on the degree of polynomial growth. The upper bound  $d_{upp}$  is equal to the total number of cycles encountered in the entire check. The lower bound  $d_{low}$  is equal to the maximum number of disjunct cycles encountered within one word during the entire check. We will give two examples to clarify the algorithm and to show the complications that can occur in the computations, which cause the use of bounds in stead of an accurate value.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{yxx, yy\}$   
We use the graph  $G_U$  in our example.

$$\widehat{xx} \rightarrow xy \leftarrow xy$$

In this case no problems occur and polynomial growth of degree 2 is detected. Algorithm 2.1 first notices that  $xxx$  contains a cycle and then tries to expand  $xx$  further. In doing so the word  $xyxy$  is encountered which also contains a cycle. Since  $|cycles| = 2$  for this word, which implies that there is a word containing 2 cycles, we have  $d_{low} = 2$ . On the other hand no other cycles are encountered, thus

$d = d_{low} = d_{upp} = 2$ . More precisely algorithm 2.1 follows the following steps.

$w = \mathbf{x}$	$V = \{\}$	$cycles = \{\}$	
$w = x\mathbf{x}$	$V = \{xx\}$	$cycles = \{\}$	
$w = xx\mathbf{x}$	$V = \{xx\}$	$cycles = \{\{1, 2\}\}$	
$w = \mathbf{x}xy$	$V = \{xx, xy\}$	$cycles = \{\{1, 1\}\}$	
$w = x\mathbf{x}yx$	$V = \{xx, xy, yx\}$	$cycles = \{\{1, 1\}\}$	
$w = xxy\mathbf{x}x$	$V = \{xx, xy, yx\}$	$cycles = \{\{1, 1\}\}$	$\mathcal{M} w$
$w = \mathbf{x}xyxy$	$V = \{xx, xy, yx\}$	$cycles = \{\{1, 1\}, \{2, 4\}\}$	
$w = x\mathbf{x}yy$	$V = \{xx, xy\}$	$cycles = \{\{1, 1\}, \{2, 2\}\}$	$\mathcal{M} w$
$w = \mathbf{x}y$	$V = \{xy\}$	$cycles = \{\}$	
$w = \mathbf{y}$	$V = \{\}$	$cycles = \{\}$	
$w = y\mathbf{y}$	$V = \{yy\}$	$cycles = \{\}$	$\mathcal{M} w, DONE$

The list *cycles* holds all pairs of indices  $\{i_1, i_2\}$  s.t.  $w[i_1 \dots i_2]$  is the beginning of a cycle. Thus  $i_2$  is sometimes lowered if  $w$  is increased. Notice that if a cycle is encountered that begins with a word  $w$  of length  $l_{\mathcal{M}}$  and this cycle is fully examined, i.e. increasing of the word means removing the cycle starting with  $w$  from the list *cycles*, we can add  $w$  to  $\mathcal{M}$ . We won't miss out on exponential growth, for this would have been detected in examining this cycle. In this way we may loose more information on the actual degree of growth though.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xxy, yy\}$

We now give an example where the degree of the polynomial growth is not detected.

$$xy \rightleftharpoons yx \rightarrow \hat{x}x$$

Algorithm 2.1 follows the following steps.

$w = \mathbf{x}$	$V = \{\}$	$cycles = \{\}$	
$w = x\mathbf{x}$	$V = \{xx\}$	$cycles = \{\}$	
$w = xx\mathbf{x}$	$V = \{xx\}$	$cycles = \{\{1, 2\}\}$	
$w = \mathbf{x}xy$	$V = \{xx\}$	$cycles = \{\{1, 1\}\}$	$\mathcal{M} w$
$w = \mathbf{x}y$	$V = \{xy\}$	$cycles = \{\}$	
$w = x\mathbf{y}x$	$V = \{xy, yx\}$	$cycles = \{\}$	
$w = xy\mathbf{x}y$	$V = \{xy, yx\}$	$cycles = \{\{1, 3\}\}$	
$w = x\mathbf{y}y$	$V = \{xy, yy\}$	$cycles = \{\{1, 1\}\}$	$\mathcal{M} w$
$w = \mathbf{y}$	$V = \{\}$	$cycles = \{\}$	
$w = y\mathbf{y}$	$V = \{yy\}$	$cycles = \{\}$	$\mathcal{M} w, DONE$

Observe that  $d_{low} = 1$  since the maximum number of disjunct cycles encountered at any stage is 1. Furthermore  $d_{upp} = 2$  since in total 2 cycles are encountered. However,  $d = 2$ . The reason that this is not detected by the algorithm is that it won't go back from a cycle to a lexicographical smaller cycle. Remember that at all times we increase words such that they satisfy equality (1) in section 1. (*end of ✦*)

### 3. COMPUTING HILBERT SERIES

We also offer a function to compute partial Hilbert Series of a monomial algebra. This Hilbert series can be used as a measurement of growth and can be used to bound Gröbner basis computations.

Given a graded algebra  $A = \bigoplus_{i=0}^{\infty} A_i$  where all the subspaces  $A_i$  are finite dimensional, define the Hilbert series by

$$H_A := \sum_{i=0}^{\infty} (\dim A_i) t^i.$$

Note that the monomial algebra we consider can always be split up into homogeneous subspaces, i.e.  $A_i$  is the subspace spanned by monomials of length  $i$ . It is immediately clear from the definition that all the coefficients of the Hilbert series will be non-negative. The function offered is an implementation of an algorithm described in [3] and it uses the concept of a graph of chains and some cohomology result.

#### 4. THE PREPROCESS-ALGORITHM

Sometimes the set of monomials  $\mathcal{M}$  can be easily *reduced*, which can save us a lot of time later on. By reduction, we mean that we can simplify the set in such a way that it doesn't effect the growth of the related monomial algebra. It will however change the set of normal words, and thus generate a different algebra. We distinguish the following two possibilities of reduction. Let  $\mathcal{A} = \{a_1, \dots, a_k\}$  and  $w \in \mathcal{A}^*$ .

- (1) if  $wa_1, \dots, wa_k \in \mathcal{M}$  then  $\mathcal{M} := (\mathcal{M} \setminus \{wa_1, \dots, wa_k\}) \cup \{w\}$
- (2) if  $a_1w, \dots, a_kw \in \mathcal{M}$  then  $\mathcal{M} := (\mathcal{M} \setminus \{a_1w, \dots, a_kw\}) \cup \{w\}$

We are only interested in the second type of reduction, which we call *left-reduction*, since the first type of reduction is standard implemented in algorithm 1.5 and 2.1 and can be achieved by mere bookkeeping, as is stated in the following remark.

**Remark 4.1.** Consider a word  $w = uv$  where  $v$  is a normal word of length  $l_{\mathcal{M}}$ . If no extension of  $v$  leads to an infinite word, then we should not consider words containing  $v$  anymore in the remainder of our search and we can achieve this by adding  $v$  to  $\mathcal{M}$ . By doing so, in the rest of the check, each word  $w' = u'v$  where  $u' > u$  will be recognized as a dead end immediately, without needing further checks.

Sometimes it is not directly obvious that left-reduction is possible. Consider the following example.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xx, xyx\}$

We can see that since  $xx$  is an obstruction, we can replace it by the monomials  $xxx, xxy$  and  $yxx$ . Now since  $xxy$  and  $yxx$  are both obstructions, we can replace them by the obstruction  $xy$ . We finally replace  $xxx$  and  $yxx$  back by  $xx$ . Thus  $\mathcal{M}$  can be left-reduced to  $\mathcal{M} = \{xx, xy\}$ .

One readily checks that if we would also have been interested in right-reduction, then by applying reduction more than once we could even reduce the set  $\mathcal{M}$  to  $\mathcal{M} = \{x\}$ , from which it would be instantly clear that  $yy\dots$  is an infinite normal word.

Thus in order to find all possibilities of left-reduction, we sometimes need to add subwords to the right of existing obstructions first. We will make this more precise.

**Definition 4.** Given an alphabet  $\mathcal{A}$  of size  $k$ . We call a set of obstructions  $\mathcal{M}$  *left-reduced* if for all obstructions  $m \in \mathcal{M}$  we have  $|\{n \in \mathcal{M} \mid n[2..|n|] \leq m[2..|m|]\}| < k$ .

Note that indeed equality would imply that left-reduction is possible; suppose  $n_i$  ( $i = 1, \dots, k$ ) are different obstructions such that  $n_i[2..|n_i|] \trianglelefteq m[2..|m|]$ . Then since  $n_i$  ( $i = 1, \dots, k$ ) are *obstructions* and thus don't divide one another,  $n_1[1], n_2[1], \dots, n_k[1]$  are all different and form the  $k$  leaves of a full subtree.

We can choose to reduce our monomial set  $\mathcal{M}$  while building the tree  $\mathcal{T}$  beforehand of algorithms 1.5 and 2.1. Reduction doesn't alter infinite paths, but only cuts down dead-ends. Therefore these algorithms, which all look for infinite paths, still work on our reduced set and return the same result.

Since left-reduction changes the group of normal words, the Hilbert series of this new algebra will differ from the original one, so we should not preprocess before computing the Hilbert series. Definition 4 hints us to a reduction-algorithm.

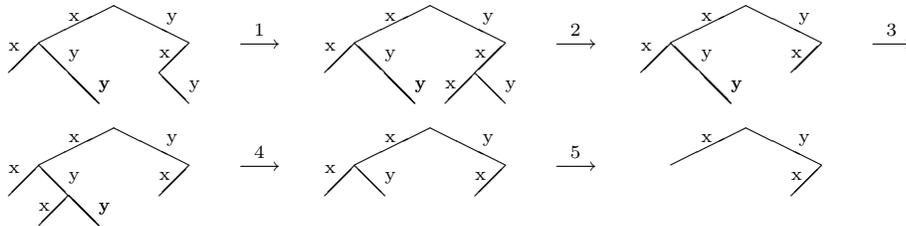
**Algorithm 4.2. - Reducing a monomial set while building the corresponding tree**

- (1) Initialize  $\mathcal{T} = []$ .
- (2) For all  $u \in \mathcal{M}$  consider all relevant expansions of  $u$ :  
i.e. for all  $v \in \mathcal{M}$  with  $|v| > |u|$  and  $v[2..|u|] = u[2..|u|]$   
consider suffix  $w$  of  $v$  of length  $|v| - |u|$   
and do:
  - Add  $uw$  to  $\mathcal{T}$ .
  - Reduce  $\mathcal{T}$  such that it doesn't contain full subtrees.
- (3) Let  $\mathcal{M}'$  be the set belonging to  $\mathcal{T}$ .
- (4) If  $\mathcal{M} \neq \mathcal{M}'$  then go to step 2 with  $\mathcal{M} := \mathcal{M}'$ .
- (5) **Return**( $\mathcal{M}$ ).

The recursive character of the algorithm is due to the fact that new possibilities of left-reduction may occur after a reduction step. This occurs in the following example.

**Example:**  $\mathcal{A} = \{x, y\}, \mathcal{M} = \{xx, xy, yxy\}$

We will show the reduction process using the tree-notation of reversed obstructions.



- (1) the extension  $xy$  of the word  $xx$  is added to the tree;
- (2) branches  $xyx$  and  $yxy$  can be reduced to branch  $xy$ ;
- (3) the extension  $xyx$  of the word  $xy$  is added to the tree;
- (4) branches  $xyx$  and  $yyx$  are reduced to branch  $yx$ ;
- (5) branches  $xx$  and  $yx$  are reduced to branch  $x$ .

Step 1 and 2 occur in the first recursion step, step 3, 4 and 5 in the second recursion step.

In stead of full preprocessing we can also choose to upper bound the number of recursions. In this way we have some control over the amount of preprocessing. Unfortunately it is unclear what amount of preprocessing could best be used in each individual case.

## REFERENCES

- [1] P.Nordbeck; *Canonical Bases for Algebraic Computations*  
Doctoral Thesis in Mathematical Sciences, LTH Lund (2001)
- [2] V.A. Ufnarovski; *A Growth Criterion for Graphs and Algebras Defined by Words*  
Mathematical Notes 31, 238-241 (1982)
- [3] V.A. Ufnarovski; *Combinatorial and Asymptotic Methods In Algebra*  
Algebra-VI, Encyclopedia of Mathematical Sciences, Volume 57, Springer (1995),5-196  
*E-mail address: C.Krook@student.tue.nl*