HOW TO DETERMINE WHETHER A PATH ALGEBRA WITH RELATIONS IS FINITE-DIMENSIONAL: THE GREEN-SØLBERG ALGORITHM

MARKUS PERLING

Abstract. We describe an algorithm due to Green and Sølberg that can determine whether the quotient of a path algebra is finite-dimensional. This algorithm is based on Gröbner bases and a mostly undocumented part of the GAP package QPA.

1. Introduction

Consider the following setup: let Q be a finite quiver (i.e. a finite directed multigraph), k some field and kQ the path algebra generated by Q over k. Such an algebra is finite-dimensional if and only if the quiver Q does not contain any directed cycles.

Let $r_1, \ldots, r_s$ be a set of polynomials in kQ and denote $\Lambda := kQ/I$ the quotient algebra by the two-sided ideal $I = \langle r_1, \ldots, r_s \rangle$. If kQ is finite-dimensional, then so is Λ. However, the converse is not true in general.

Given a quiver with cycles, to determine whether a set of relations yields a finite-dimensional quotient algebra can be a tricky business.

Example 1.1: Consider the following quiver:

$$\begin{tikzcd}
1 \arrow[loop left, "w"] \arrow[r, bend right, "x"'] & 2 \arrow[l, bend right, "y"'] \arrow[loop right, "z"]
\end{tikzcd}$$

and the following relations: $w^3, (xy)^3, z^3$. As these relations are monomial, they already form a Gröbner basis. Obviously, the algebra $kQ/\langle w^3, (xy)^3, z^3 \rangle$ is not finite-dimensional, as, e.g. for $n \geq 1$, $(xzy)^n$ is not divisible by any of these monomials . We could now proceed by adding more relations of this kind, such as $(xzy)^3, (xz^2y)^3, (xzzyw)^3, \ldots$.

Now consider the following cycles $a := xzy$, $b := xz^2y$ and their compositions

$$abbabaabbaababbaba\ldots$$

following the prefixes of the Thue-Morse sequence [Slo, A010060]. This sequence is known to contain no triple repetitions of any pattern of any length. As any such prefix represents a cycle in Q, this implies that we cannot make Λ finite-dimensional by uniformly adding relations of the type c^3 for cycles c in Q.

Of course, we can eliminate these Thue-Morse prefixes by simply adding a or b to the relations. It would be interesting to have an explicit understanding what are the necessary cycle cancellations needed to ensure finite-dimensionality.

In this note we will describe an algorithm due to Green and Sølberg that determines whether Λ is finite-dimensional for any given quiver with relations for which a Gröbner basis can be computed. This algorithm has been implemented in the GAP package QPA [GS$^+$24]. For general reference on Gröbner bases, we refer to [FFG93], [Kel97]¸ [Gre99], [Gre00].

We will assume throughout that, given some monomial well-ordering on kQ, we can compute and fix a *reduced* Gröbner basis $g_1, \ldots, g_t$ for I. For any polynomial p in kQ, we denote $\textsc{lm}(p)$ its lead monomial. When no ambiguity arises, we will identify monomials in kQ with their corresponding paths in Q. In particular, any directed cycle in Q corresponds to a monomial in kQ. We also consider loops (i.e. arrows that start and end at the same node) as directed cycles.

This is our first observation:

Lemma 1.2: *Given a Gröbner basis $g_1, \ldots, g_t$ for I, then the set of monomials in KQ that are not divisible by any $\textsc{lm}(g_i)$ represent a basis of Λ. In particular, Λ is finite-dimensional if and only if there exist only finitely many monomials in kQ that are not divisible by some $\textsc{lm}(g_i)$.*

Proof. Denote $\{b_i \mid i \in A\}$ (for some index set A) monomials in kQ that represent a basis for Λ. Recall (e.g. [Gre99, §2.2.3]) that $\langle b_i \mid i \in A\rangle_k = \langle \textsc{NonTip}(I)\rangle_k$, and $KQ = I \oplus \langle \textsc{NonTip}(I)\rangle_k$ as k-vector spaces. This splitting is compatible with the choice of the set of monomials of kQ as basis for kQ as a k-vector space. Then by using Buchberger reduction we can bring any monomial into the form:

$$c = \sum_{i=1}^{t} f_i g_i h_i + \sum_{j \in A}^{\text{finite}} \alpha_j b_j,$$

where $\alpha_j \in k$ and $f_i, h_i \in kQ$. Because the g_i are a Gröbner basis for I, the remainder of c modulo the g_i is unique and therefore the α_j are.

If at least one of the terms $f_i g_i h_i$ is nonzero, then at least one Buchberger reduction step has been performed, hence $\textsc{lm}(g_i)$ divides c for at least one i.

If all terms $f_i g_i h_i$ are zero, then c is not divisible by any $\textsc{lm}(g_i)$ and therefore coincides with one of the b_j.

It follows that the complete reduction algorithm yields a unique representative for each nonzero monomial in Λ such that none of the representative's term's are divisible by any $\textsc{lm}(g_i)$. Equivalently, we obtain a natural isomorphism of k-vector spaces $\Lambda \simeq \langle \textsc{NonTip}(I)\rangle_k$ and the lemma follows. $\square$

With this observation, we can rephrase the problem of determining the finite-dimensionality of Λ as follows:

Given a quiver Q and a Gröbner basis $g_1, \ldots, g_t$ for I, can we determine whethere there are only finitely many monomials in kQ that are not divisible by any $\textsc{lm}(g_i)$?

The Green-Sølberg algorithm very elegantly solves this problem by using dictionary trees (tries). Here, we consider each label in the quiver Q (including both the labels for arrows and idempotents) as a symbol in a formal alphabet and a monomial in Q as a word of that alphabet.

Given a list of words, such as $\textsc{lm}(g_1), \ldots, \textsc{lm}(g_t)$, it is natural to consider dictionary data structures that efficiently support basic path algebra arithmetics. For example, division with remainder with respect to a Gröbner basis generally involves a great deal of substring searches. It has been proposed by Benjamin Keller [Kel97] to use tree-like data structures for this purpose, so-called tries.

So, the Green-Sølberg algorithm performs the following steps:

(1) From a given Gröbner basis $\mathcal{G} = \{g_1, \ldots, g_t\}$, construct a dictionary trie $T_{\mathcal{G}}$ (more precisely, a trie with failure nodes, as used in the Aho-Corasick algorithm [AC75]).
(2) From $T_{\mathcal{G}}$ construct a directed multigraph $S_{\mathcal{G}}$ that contains all the non-root nodes of $T_{\mathcal{G}}$ and a distinguished source node $\circ$.
(3) We consider $S_{\mathcal{G}}$ as a *state machine* where $\circ$ is the single start node and any other node is a possible stop node. Proposition 3.1 will show that the words generated by $S_{\mathcal{G}}$ coincide with the monomials of kQ.
(4) In a finite step, the truncated state machine $S_{\mathcal{G},tr}$ is obtained by removing the nodes of $S_{\mathcal{G}}$ corresponding to the leaf nodes of $T_{\mathcal{G}}$.
(5) Theorem 3.2 will show that the words generated by $S_{\mathcal{G},tr}$ coincide with a monomial basis for Λ. In particular, Λ is finite-dimensional if and only if $S_{\mathcal{G},tr}$ has no directed cycles.

Veryfing whether $S_{\mathcal{G},tr}$ has oriented cycles can be done e.g. by straightforward depth-first search in the underlying multigraph. Other uses of $S_{\mathcal{G},tr}$ are:

- It can produce a full monomial basis of Λ if $S_{\mathcal{G},tr}$ has no directed cycles.
- It can produce partial monomial bases of Λ, e.g. by bounding the length of words, regardless whether $S_{\mathcal{G},tr}$ has cycles or not.
- Using standard algorithms such as Johnson's [Joh75], we can detect all remaining irreducible cycles in Λ.

In Section 2 we will give a number of explicit examples for constructing $T_{\mathcal{G}}$, $T_{\mathcal{G}}$, and $S_{\mathcal{G},tr}$. In Section 3 we will fill out the details of the Green-Sølberg algorithm as outlined above.

2. Examples of quivers, tries, and state machines

For simplicity, the examples in this section will only deal with monomial relations. As a set of monomials that are not mutually divisible already constitutes a reduced Gröbner basis for the ideal they generate, this will save us any discussion of well-orderings and Gröbner basis computations.

A crucial part of the algorithm is that we consider monomial normalized with respect to leading idempotents. That is, whenever we consider a word in a trie or state machine, we (implicitly) assume that it is prefixed by its source idempotent with multiplicity 1. E.g. in Example 2.1, we have the monomials $e_1, x, x^2, \ldots$ as a basis of kQ, but whenever we add some x^i into a trie, it will be normalized to $e_1 x^i$. Likewise, we will always assume that interior or ending idempotents are being absorbed, e.g. we will never consider words of the form $e_1 x^i e_1^j x^k e_1^l$ with $j, k > 0$. However, to simplify presentation, if the monomial is not idempotent, we will almost always suppress the leading idempotent, e.g. write $e_1, x, x^2, \ldots$ instead of $e_1, e_1x, e_1x^2, \ldots$.

Example 2.1: Consider the quiver

$$\begin{tikzcd}
1 \arrow[loop left, "x"']
\end{tikzcd}$$

and the relation x^3. Then the monomial base of Λ is given by e_1, x, x^2. A trie without failure nodes containing the single word $e_1x^3 = x^3$ looks like this:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[d, "x"'] \\
x \arrow[d, "x"'] \\
xx \arrow[d, "x"'] \\
xxx
\end{tikzcd}$$

Removing the leaf node 1_{xxx}, we get:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[d, "x"'] \\
x \arrow[d, "x"'] \\
xx
\end{tikzcd}$$

Considered as a state machine with start node $\circ$ and any stop node except $\circ$, it can generate the following paths:

$$e_1, e_1x, e_1x^2.$$

Absorbing e_1, we get a little more conveniently: e_1, x, x^2.

Example 2.2: Now consider

$$\begin{tikzcd}
1 \arrow[loop left, "x"] \arrow[loop right, "y"]
\end{tikzcd}$$

These are the dictionary tries when successively adding x^2, y^2, xy:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[d, "x"'] \\
x \arrow[d, "x"'] \\
xx
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[d, "x"'] & & y \arrow[d, "y"'] \\
xx & & yy
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[d, "x"'] \arrow[dr, "y"] & & y \arrow[d, "y"'] \\
xx & xy & yy
\end{tikzcd}$$

As we can see, none of these tries represent all the nonzero monomials in the quotient algebra. For these, we must make use of another feature of tries: *failure nodes*. That is, for each prefix in a trie (i.e. each string represented by a node), its failure node is the node corresponding to the longest proper suffix of that string (normalized wrt. leading idempotent). E.g., the longest proper suffix of e_1x^2 is $x = e_1x$, that of e_1xy is $y = e_1y$, and that of y is e_1. The resulting failure nodes are indicated by the dashed lines:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[d, "x"'] \\
x \arrow[d, "x"'] \arrow[u, dashed, bend right] \\
xx \arrow[u, dashed, bend right]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[d, "x"'] \arrow[ur, dashed, bend left] & & y \arrow[d, "y"'] \arrow[ul, dashed, bend right] \\
xx \arrow[u, dashed, bend right] & & yy \arrow[u, dashed, bend right]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[d, "x"'] \arrow[dr, "y"] \arrow[ur, dashed, bend left] & & y \arrow[d, "y"'] \arrow[ul, dashed, bend right] \\
xx \arrow[u, dashed, bend right] & xy \arrow[ur, dashed] & yy \arrow[u, dashed, bend right]
\end{tikzcd}$$

These failure nodes facilitate the completion of the trie into a directed multigraph that, considered as a state machine with starting node $\circ$ and arbitrary end nodes, produces the same paths as the original quiver. Given any node in the trie representing a path p, we have a number (possibly zero) of outgoing arrows labeled $a_{i_1}, \ldots, a_{i_k}$, being a subset of all possible letters $a_1, \ldots, a_k$ such that pa_i for $1 \leq i \leq k$ is a valid path in the original quiver. So, the idea now is to top-down add arrows as follows:

Denote f the failure node of p.

- If f is undefined (equivalently, $p = e_i$ for some i), then for each a_l that is not one of the a_{i_k}, add an arrow labelled a_l starting at e_i and ending at the node e_j, where j is the target of a_l in the original quiver.
- If $f = i$ for some i, then for each a_l that is not one of the a_{i_k}, add an arrow labelled a_l starting at e_i and ending at the node $e_j j$, where j is the target of a_l in the original quiver.
- Otherwise, for each a_i that is not one of the a_{i_k}, add an arrow labelled a_i starting at p and ending at the target of the arrow starting at f and labelled a_i.

The existence of an arrow starting at f and labelled a_i is guaranteed by going top-down. Here are the resulting graphs:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[loop right, "y"] \arrow[d, "x"] \\
x \arrow[u, bend left, "y"] \arrow[d, "x"'] \\
xx \arrow[loop right, "x"] \arrow[uu, bend left=60, "y"]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[rr, bend left=10, "x"] \arrow[d, "x"'] & & y \arrow[ll, bend left=10, "y"] \arrow[d, "y"] \\
xx \arrow[loop left, "x"] \arrow[urr, "y"] & & yy \arrow[loop right, "y"] \arrow[ull, "x"']
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[d, "x"'] \arrow[dr, bend right, "y"'] & & y \arrow[ll, "x"'] \arrow[d, "y"] \\
xx \arrow[loop left, "x"] \arrow[r, "y"'] & xy \arrow[ul, "x"] \arrow[r, "y"'] & yy \arrow[loop right, "y"] \arrow[ull, "x"']
\end{tikzcd}$$

It is straightforward to see that starting from $\circ$, each of these graphs produces all possible paths as in the original quiver. These graphs therefore necessarily contain directed cycles. Now, to obtain graphs that can produce all nonzero paths of the quotient algebra, we simply remove the leaf nodes of the original trie:

$$\begin{tikzcd}
\circ \arrow[d, "e_1"'] \\
e_1 \arrow[loop right, "y"] \arrow[d, "x"] \\
x \arrow[u, bend left, "y"]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x \arrow[rr, bend left=10, "x"] & & y \arrow[ll, bend left=10, "y"]
\end{tikzcd}
\qquad
\begin{tikzcd}
& \circ \arrow[d, "e_1"'] & \\
& e_1 \arrow[dl, "x"'] \arrow[dr, "y"] & \\
x & & y \arrow[ll, "x"']
\end{tikzcd}$$

We see that these graphs produce the path bases of $kQ/\langle x^2\rangle$, $kQ/\langle x^2, y^2\rangle$, and $kQ/\langle x^2, y^2, xy\rangle$, respectively, where only the last one has no directed cycles.

Example 2.3: Consider again:

$$\begin{tikzcd}
1 \arrow[loop left, "w"] \arrow[r, bend right, "x"'] & 2 \arrow[l, bend right, "y"'] \arrow[loop right, "z"]
\end{tikzcd}$$

with relations

$$w^3, wxyw^2, z^3.$$

These relations yield the following trie, with the failure nodes indicated by dotted arrows.

$$\begin{tikzcd}
& & \circ \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & & \\
& e_1 \arrow[d, "w"] & & e_2 \arrow[d, "z"] & \\
& w \arrow[u, dashed, bend left] \arrow[dl, "w"'] \arrow[dr, "x"] & & z \arrow[u, dashed, bend right] \arrow[d, "z"] & \\
ww \arrow[ur, dashed, bend left] \arrow[d, "w"] & & wx \arrow[ur, dashed] \arrow[d, "y"] & zz \arrow[u, dashed, bend right] \arrow[d, "z"] & \\
www \arrow[u, dashed, bend left] & & wxy \arrow[uuul, dashed, bend right] \arrow[d, "w"] & zzz \arrow[u, dashed, bend right] & \\
& & wxyw \arrow[uuul, dashed, bend left] \arrow[d, "w"] & & \\
& & wxyww \arrow[uuull, dashed] & &
\end{tikzcd}$$

The same procedure as in the previous example yields the following directed graph that produces precisely a basis for in the infinite-dimensional quotient algebra Λ:

$$\begin{tikzcd}
 & & \circ \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & \\
 & e_1 \arrow[rr, bend right, "x"'] \arrow[d, "w"'] & & e_2 \arrow[ll, bend right, "y"'] \arrow[d, "z"] \\
 & w \arrow[dl, "w"'] \arrow[dr, "x"] & & z \arrow[ull, "y"] \arrow[d, "z"] \\
ww \arrow[rr, "x"] & & wx \arrow[ur, "z"] \arrow[d, "y"'] & zz \arrow[uull, "y"] \\
 & & wxy \arrow[uuur, "x"] \arrow[d, "w"'] & \\
 & & wxyw \arrow[uu, bend left, "x"] &
\end{tikzcd}$$

Example 2.4: Consider again:

$$\begin{tikzcd}
1 \arrow[loop left, "w"] \arrow[r, bend right, "x"'] & 2 \arrow[l, bend right, "y"'] \arrow[loop right, "z"]
\end{tikzcd}$$

with relations w^2, xy, z^2, xzy. The resulting state machine is:

$$\begin{tikzcd}
 & & \circ \arrow[dl, "e_1"'] \arrow[dr, "e_2"] & \\
 & e_1 \arrow[dl, "w"'] \arrow[d, "x"'] & & e_2 \arrow[ll, "y"'] \arrow[d, "z"'] \\
w \arrow[r, "x"'] & x \arrow[d, "z"'] & & z \arrow[ull, "y"] \\
 & xz & &
\end{tikzcd}$$

It is cycle free and generates the following paths:

$$e_1, e_2, w, wx, wxz, x, xz, z, zy, zyw, zywx, zywxz, zyx, zyxz, y, yw, ywx, ywxz, yx, yxz.$$

So $\dim \Lambda = \dim kQ/\langle w^2, xy, z^2, xzy\rangle = 20$.

3. The algorithm

We are given a quiver Q, the path algebra kQ, and a *reduced* Gröbner basis $\mathcal{G} = \{g_1, \ldots, g_t\}$ for some ideal I in kQ. We will mostly use the terms *path*, *monomial*, and *word* interchangeably. However, as in the previous section, we assume that prefixes are normalized with respect to leading idempotents. That is, given a path p, we consider the associated prefix to be of the form $\text{source}(p)p'$, where p' does not contain any symbol corresponding to a node in Q, and $p = p'$ as monomials in kQ. We will implicitly assume that any subword of a given p will be normalized like this when considered as a prefix.

Following the outline given in the introduction, we start with describing the trie data structure $T_{\mathcal{G}}$. We will deviate a little from the list-based implementation in QPA and for simplicity we will represent the trie using `dict` data structures, as are available in the standard libraries of many programming languages such as Julia or Python.

Thus $T_{\mathcal{G}}$ consists of key-value pairs, where the keys are the prefixes in $T_{\mathcal{G}}$. For a given prefix p, its associated value $T_{\mathcal{G}}[p]$ is a (possibly empty) set $\{a_1, \ldots, a_k\}$ such that $pa_1, \ldots, pa_k$ are precisely all prefixes in $T_{\mathcal{G}}$ that extend p by one letter.

In parallel, using another dict `dict` $F_{\mathcal{G}}$, we keep track of *failure nodes* (or suffix links, respectively). For some idempotent normalized prefix p of length > 1, $F_{\mathcal{G}}[p]$ returns the idempotent normalized prefix *source*$(q)q$ in $T_{\mathcal{G}}$ where q represents longest *proper suffix* of p that is available in $T_{\mathcal{G}}$. We use the conventions that $\emptyset$ always is a proper suffix, and for an idempotent normalized prefix source$(p)p$, p is *not* a proper suffix.

In the algorithms described below, we use the following helper functions:

- head(p) returns the first symbol in a word p.
- tail(p) returns the remaining word of p after the first symbol.
- suffix(p, k) returns the suffix of the word p of length k.
- append(p, a) for a prefix p and an arrow a returns the word pa.
- We do not use any prepend function, i.e. for the concatination of an idempotent e symbol and a path p we will write ep.

We initialize $T_{\mathcal{G}}$ such that $T_{\mathcal{G}}[\emptyset] := \{e_i \mid i \in Q_0\}$. Algorithm 1 shows the trie insertion procedure INSERT that is applied to each monomial LM(g_i).

Algorithm 1 Insert path into $T_{\mathcal{G}}$

```
Input: An idempotent normalized path $p$ in $Q$ and $T_{\mathcal{G}}$.
1: procedure INSERT($p$, $T_{\mathcal{G}}$)
2:     $prefix := \mathrm{head}(p)$

3:     while $\mathrm{length}(p) > 0$ do
4:         if $prefix \notin \mathrm{keys}(T_{\mathcal{G}})$ then
5:             $T_{\mathcal{G}}[prefix] := \emptyset$

6:         $p := \mathrm{tail}(p)$

7:         if $\mathrm{length}(p) > 0$ then
8:             $arrow := \mathrm{head}(p)$

9:             if $arrow \notin \mathrm{T}_{\mathcal{G}}[prefix]$ then
10:                $T_{\mathcal{G}}[prefix] := T_{\mathcal{G}}[prefix] \cup \{arrow\}$

11:            $prefix := \mathrm{append}(prefix, arrow)$
```

After all LM(g_i) have been inserted into $T_{\mathcal{G}}$, we initialize $F_{\mathcal{G}}$ as an empty `dict`. Then, doing a *breadth-first* traversal, we call

$$\textsc{createFailureNode}(p, T_{\mathcal{G}}, F_{\mathcal{G}})$$

(see Algorithm 2) for each prefix p in $T_{\mathcal{G}}$.

Next, Algorithm 3 creates the state machine $S_{\mathcal{G}}$ from $T_{\mathcal{G}}$ and $F_{\mathcal{G}}$. We choose to represent $S_{\mathcal{G}}$ as a `dict` whose keys are prefixes and values are sets of pairs (symbol, target), where symbol $\in Q_0 \cup Q_1$ and target is a prefix. For sake of exposition, we have earlier denoted the starting node in $S_{\mathcal{G}}$ by $\circ$, which we should consider as an alias for $\emptyset$, i.e. let $\circ := \emptyset$.

Algorithm 2 Create failure node

```
Input: An idempotent normalized path $p$ in the quiver $Q$, a complete trie $T_{\mathcal{G}}$ for $\mathcal{G}$, and $F_{\mathcal{G}}$.
 1: procedure CREATEFAILURENODE($p$, $T_{\mathcal{G}}$, $F_{\mathcal{G}}$)
 2:     if $length(p) > 1$ then
 3:         $F_{\mathcal{G}}[p] := target(p)$                    ▷ Default when no proper suffix in $T_{\mathcal{G}}$

 4:     suffixLength := 1

 5:     while suffixLength < length($p$) − 1 do           ▷ Proper suffix after idempotent absorption
 6:         sfx := suffix($p$, suffixLength)
 7:         sfx := source(sfx) sfx                         ▷ Idempotent normalization

 8:         if sfx $\in$ keys($T_{\mathcal{G}}$) then
 9:             $F_{\mathcal{G}}[p]$ := sfx

10:         suffixLength := suffixLength +1
```

Algorithm 3 Create state machine $S_{\mathcal{G}}$ from $T_{\mathcal{G}}$ and $F_{\mathcal{G}}$

```
Input: A trie $T_{\mathcal{G}}$ and its failure nodes $F_{\mathcal{G}}$.
Output: A state machine $S_{\mathcal{G}}$
 1: function CREATESTATEMACHINE($T_{\mathcal{G}}$, $F_{\mathcal{G}}$)
 2:     $S_{\mathcal{G}}$ := an empty dict.

 3:     for $p \in$ keys($T_{\mathcal{G}}$) $\setminus \{\emptyset\}$ do                ▷ Initialize nodes of $S_{\mathcal{G}}$
 4:         $S_{\mathcal{G}}[p] := \emptyset$

 5:     $S_{\mathcal{G}}[\emptyset] := \{(e_i, e_i) \mid e_i \in Q_0\}$                  ▷ Set successors of start node

 6:     for $e_i \in Q_0$ do                ▷ Special initialization procedure for top level nodes.
 7:         for each arrow $a$ in $Q_1$ with source $a = e_i$ do
 8:             if $e_i a \in$ keys($T_{\mathcal{G}}$) then
 9:                 $S_{\mathcal{G}}[e_i] := S_{\mathcal{G}}[e_i] \cup \{(a, e_i a)\}$
10:             else
11:                 $S_{\mathcal{G}}[e_i] := S_{\mathcal{G}}[e_i] \cup \{(a, \text{target}(a))\}$

12:     trieDepth := max{length($p$) | $p \in$ keys($T_{\mathcal{G}}$)}

13:     for $i = 2$ to trieDepth do ▷ This is a very naive breadth-first implementation - improve at will.
14:         for $p \in$ keys($S_{\mathcal{G}}$) do
15:             if length($p$) $= i$ then
16:                 for $a \in Q_1$ with source($a$) = target($p$) do
17:                     if $a \in T_{\mathcal{G}}[p]$ then
18:                         $S_{\mathcal{G}}[p] := S_{\mathcal{G}}[p] \cup \{(a, \text{append}(p, a))\}$
19:                     else
20:                         $S_{\mathcal{G}}[p] := S_{\mathcal{G}}[p] \cup \{(a, \text{append}(F_{\mathcal{G}}[p], a))\}$

21:     return $S_{\mathcal{G}}$
```

Note that by construction in Algorithm 3, the nodes of $T_{\mathcal{G}}$ are a subset of those of $S_{\mathcal{G}}$. The latter contains all prefix nodes of $T_{\mathcal{G}}$ and additionally all e_i that are not the source of any LM(g_i). We call a node in $S_{\mathcal{G}}$ a *leaf* node if it is a leaf node in $T_{\mathcal{G}}$. The following summarizes the most important properties of $S_{\mathcal{G}}$:

Proposition 3.1: *(i) The words generated by the state machine $S_\mathcal{G}$ returned by Algorithm 3 are in one-to-one correspondence with the monomials in kQ.*

(ii) For any monomial in kQ that is divisible by some $\text{LM}(g_i)$, the corresponding word in the state machine $S_\mathcal{G}$ passes through a leaf node.

(iii) Conversely, if a monomial is not divisible by any $\text{LM}(g_i)$, then corresponding word in the state machine $S_\mathcal{G}$ does not pass through a leaf node.

Proof. (i) We refer to line numbers of Algorithm 3. By line 5, we see that $S_\mathcal{G}$ generates all idempotents. Also, these are the only arrows labelled by the e_i, hence the assertion is true for monomials e_i. Now, in the loop of lines 6 and 13, for any prefix p (including the e_i), the node in $S_\mathcal{G}$ labelled p has the same number of outgoing arrows as the node target p in Q_0, with the same labels. It follows that any monomial in kQ can be represented by a unique word of $S_\mathcal{G}$. So the assertion follows.

(ii) Let p be any monomial that factorizes as $a\text{LM}(g_j)b$ for some j, where without loss of generality assume that a is not divisible by any $\text{LM}(g_i)$. If the stop node of a in $S_\mathcal{G}$ is source(g_j) then p will pass through the leaf node of $\text{LM}(g_j)$ and the assertion holds. Otherwise, denote q the prefix that represents the stop node of a. Moreover, denote $\text{LM}(g_ij = c_1 \cdots c_k$ the arrow decomposition of $\text{LM}(g_j)$. Then the target of c_1 starting at q is a prefix that ends in c_1. We claim that for any $1 \leq l \leq k$ the stop node of $ac_1 \cdots c_l$ corresponds to a prefix that has a suffix $c_1 \cdots c_l$. Then for $l = k$ we conclude by the reducedness of $\mathcal{G}$ that the stop node of $a\text{LM}(g_i)$ coincides with that of $\text{LM}(g_i)$, hence is a leaf node.

We have proven the claim for $l = 1$. By induction, for $1 \leq l < k$, assume that the stop node of the word $ac_1 \cdots c_l$ is some $q_l c_1 \cdots c_l$. Then the stop node of $ac_1 \cdots c_{l+1}$ is of the form $q'c_{l+1}$, where q' is the longest proper suffix of $q_l c_1 \cdots c_l$ that is a prefix in $T_\mathcal{G}$. By the induction assumption, the set of prefixes that contain a suffix of $q_l c_1 \cdots c_l$ contains the prefix $c_1 \cdots c_l$ (which is a prefix of $\text{LM}(g_i)$ and therefore in $T_\mathcal{G}$). Hence $q'c_{l+1}$ indeed must be of the form $q_{l+1}c_1 \cdots c_{l+1}$ and the assertion follows.

(iii) Let p be a monomial that is not divisible by any $\text{LM}(g_i)$. First we show that if p shares a prefix with any of the $\text{LM}(g_i)$, then without loss of generality we can replace p by a proper suffix. Denote by q the longest prefix of p $T_\mathcal{G}$. As p is not divisible by any $\text{LM}(g_i)$, q does not coincide with one of the $\text{LM}(g_i)$, hence q does not pass through any leaf node. If we write $p = qaq'$, where a is a single arrow or one of the e_i, and q' is possibly empty, then the state of qa in $S_\mathcal{G}$ is represented by a path ra, where r is a proper suffix of q. Therefore, if p passes through a leaf node, then so does $p' := raq'$. So we can replace p by p'. Repeating this replacement a finite number of times, we obtain a polynomial q that share no prefix with any of the $\text{LM}(g_i)$ and, if p passes throught a leafe node, then so does q.

Next, we show that without loss of generality we can assume that p does share a prefix with one of the $\text{LM}(g_i)$. Denote $p = p'p''$, where q'' is the longest (possibly empty) proper suffix such that target$(p')p''$ shares a prefix with one of the $\text{LM}(g_i)$. Then p' cannot pass through a leaf node. Hence, if p passes through a leaf node, then so does p''. So we can replace p by p''.

Each of these replacements produces each time a shorter path and can always be applied to any nonempty path p whose corresponding monomial is not divisible by any of the $\text{LM}(g_i)$. This implies that without loss of generality we can assume that the path p is empty. Hence, p cannot pass through a leaf node. $\square$

Now, as a final step we want to apply these results to understand the quotient ring Λ. By 3.1(i), the words generated by $S_\mathcal{G}$ naturally be identified with the basis of kQ as a k-vector space. By Lemma 1.2 and 3.1(ii) & (iii), the monomials in $\langle \text{NonTip}(I) \rangle_k$ are one-to-one with words of $S_\mathcal{G}$ that do not factor through a leaf node of $S_\mathcal{G}$. So, in order to only generate these monomials, we can use a smaller state machine with the leaf nodes removed.

Definition: We construct the *truncated* state machine $S_{\mathcal{G},tr}$ from $S_\mathcal{G}$ by removing its leaf nodes and arrows starting or ending at leaf nodes.

We now obtain the following theorem:

Theorem 3.2: *Let Q be a finite quiver, kQ its path algebra over a field k, I an ideal in kQ, $\mathcal{G}$ a reduced Gröbner basis of I, and $S_{\mathcal{G},tr}$ the truncated state machine obtained from $\mathcal{G}$. Then the words generated by $S_{\mathcal{G},tr}$ can naturally be identified with a monomial basis of kQ/I. Moreover, kQ/I is finite-dimensional if and only if $S_{\mathcal{G},tr}$ has no directed cycles.*

Remark 3.3: Note that from the assumption that $\mathcal{G}$ is reduced, it follows that a leaf node can never be a failure node in $F_{\mathcal{G}}$. Therefore, instead of first adding leaf nodes to $S_{\mathcal{G}}$ and then remove them to get $S_{\mathcal{G},tr}$, it is equivalent and more efficient to skip the creation of leaf nodes in Algorithm 3. This can be achieved by replacing line 6 with

```
**if** $e_i \in Q_0$ and $e_i$ is not a leaf in $T_{\mathcal{G}}$ **then**
```

. line 13 with

```
**for** $i = 2$ to trieDepth $-1$ **do**
```

. and line 15 with

```
**if** length$(p) = i$ and $p$ is not a leaf in $T_{\mathcal{G}}$ **then**
```

The output of Algorithm 3 will then be $S_{\mathcal{G},tr}$.

4. Some remarks on the use of tries

The use of tries as outlined in Section 3 relies on incorporating idempotent normalization, which in turn requires a somewhat custom tailored trie implementation. In practice, it may be preferable to use a general purpose trie implementation. Let us make the assumption that there is no idempotent e_i among the $\textsc{lm}(g_1), \dots, \textsc{lm}(g_t)$. Then we can produce a trie $T'_{\mathcal{G}}$ and failure nodes $F'_{\mathcal{G}}$ using text book functions insert and createFailureNode for trie creation. Assuming that these functions operate on the same `dict` data structures as ours, the outcome will be:

- $T'_{\mathcal{G}}[\emptyset] = \{\text{head}(\textsc{lm}(g_i)) \mid g_i \in \mathcal{G}\}$, keys$(T'_{\mathcal{G}})$ does not contain any e_i and all other keys of $T'_{\mathcal{G}}$ coincide with that of $T_{\mathcal{G}}$ with absorbed idempotents.
- For any nonempty path p in $T'_{\mathcal{G}}$, we have either $F'_{\mathcal{G}}[p] = \emptyset$ if $F_{\mathcal{G}}[\text{source}(p)p] = e_i$ for some i, or otherwise $F'_{\mathcal{G}}[p] =$ the idempotent absorption of $F_{\mathcal{G}}[\text{source}(p)p]$.

With this outcome, to call Algorithm 3 with $T'_{\mathcal{G}}$ and $F'_{\mathcal{G}}$, we can modify it as follows. We replace line 4 with:

```
$S_{\mathcal{G}}[\text{source}(p)p] := \emptyset$
```

and lines 13 to 20 with:

```
**for** $i = 1$ to trieDepth **do**
    **for** $p \in$ keys$(S_{\mathcal{G}})$ **do**
        **if** length$(p) = i + 1$ **then**
            **for** $a \in Q_1$ with source$(a) =$ target$(p)$ **do**
                **if** $a \in T_{\mathcal{G}}[p]$ **then**
                    $S_{\mathcal{G}}[p] := S_{\mathcal{G}}[p] \cup \{(a, \text{append}(p, a))\}$
                **else**
                    $f := F_{\mathcal{G}}[p]$

                    **if** $f = \emptyset$ **then**
                        $f :=$ target$(p)$
                    **else**
                        $f :=$ source$(f)f$

                    $S_{\mathcal{G}}[p] := S_{\mathcal{G}}[p] \cup \{(a, \text{append}(f, a))\}$
```

The optimizations mentioned in Remark 3.3 can be incorporated in the obvious way.

References

[AC75] A. Aho and M. J. Corasick, *Efficient string matching: an aid to bibliographic search*, Communicatinos of the ACM **18** (1975), no. 6, 333–340.

[FFG93] D. R. Farkas, C. D. Feustel, and E. L. Green, *Synergy in the theories of Gröbner bases and path algebras*, Can. J. Math. **45** (1993), no. 4, 727–739.

[Gre00] E. L. Green, *Multiplicative bases, Gröbner bases, and right Gröbner bases*, J. Symb. Comp. **29** (2000), 601–623.

[Gre99] ———, *Noncommutative Gröbner bases, and projective resolutions*, Computational methods for representations of groups and algebras, 1999, pp. 29–60.

[GS$^+$24] E. L. Green, O. Sølberg, et al., *GAP Package QPA*, 2024. Version 1.35, `https://www.gap-system.org/Packages/qpa.html`.

[Joh75] D. B. Johnson, *Finding all the elementary circuits of a directed graph*, SIAM J. Comput. **4** (1975), no. 1, 77–84.

[Kel97] B. Keller, *Algorithms and Orders for Finding Noncommutative Gröbner Bases*, Ph.D. Thesis, Virginia Polytechnic Institute and State University, 1997.

[Slo] N. J. A Sloane, *The Online Encyclopedia of Integer Sequences.* `http://oeis.org`.