MRC*– A System for Computing Gröbner Bases in Monoid and Group Rings

Birgit Reinert† and Dirk Zeckzer
Universität Kaiserslautern, 67663 Kaiserslautern
{reinert,zeckzer}@informatik.uni-kl.de

Abstract

Gröbner bases and Buchberger's algorithm have been generalized to monoid and group rings. This paper presents a discription of an implementation of prefix Gröbner basis procedures in this setting.

1 Introduction

In 1965 Buchberger introduced the theory of Gröbner bases for ideals in commutative polynomial rings over fields [4], which allows solving many problems related to polynomial ideals in a computational fashion using rewriting methods. The most familiar problem is the ideal membership problem, i.e. the problem of deciding whether a given polynomial lies in an ideal specified by a generating set. In case the generating set is a Gröbner basis this problem becomes solvable by checking whether the polynomial reduces to zero with the computed Gröbner basis.

Nowadays implementations of Buchberger's algorithm for computing Gröbner bases are provided by all major computer algebra systems. The importance of Gröbner bases for the study of ideals has led to generalizations of the Gröbner basis theory for non-commutative structures. For example, originating from special problems in physics, Lassner in [7] suggests how to extend existing computer algebra systems in order to handle special classes of non-commutative algebras, e.g. Weyl algebras and later on Lie algebras [2]. This works because the structures studied allow representations by *commutative* polynomials for their elements. Unfortunately this approach is not possible for general non-commutative algebras. Mora [14] generalized Gröbner bases for non-commutative polynomial rings, i.e. free monoid rings, where multiplication of terms is just concatenation of words. An implementation of his procedure has for example been done in the system OPAL by Keller [6].

The next step of generalizing Gröbner bases was to study arbitrary monoid and group rings. The theoretical background on prefix Gröbner bases was first presented at ISSAC'93 by Madlener and Reinert in [9]. Different specialized definitions of reduction relations followed [15, 11, 10, 12]. In [13] connections between the word problem for monoids and groups and the ideal membership problem in free monoid and free group rings, respectively, as well as connections between the submonoid problem and the subalgebra problem and between the subgroup problem and the one-sided ideal membership problem are proven. [16] shows how the well-known procedure of Todd and Coxeter for enumerating cosets of a subgroup in a given group can be implemented using

*MRC (pronounce MaRCie) stands for **M**onoid **R**ing **C**ompletion.

†The author was supported by the Deutsche Forschungsgemeinschaft (DFG).

prefix Gröbner bases (in fact the implementation used to compute the examples there was done in MRC). Prefix Gröbner bases in commutative polynomial rings are in fact special Janet bases. Here we want to present an implementation of the Gröbner basis method using *prefix reduction* in monoid and group rings over the rationals $\mathbb{Q}$. An extension of the system to cover Gröbner basis methods using commutative reduction (for commutative monoids) and quasi-commutative reduction (for polycyclic groups) is planned.

The paper is organized as follows: Section 2 summarizes the theoretical results and the procedures implemented. Section 3 introduces the system MRC and provides an example to illustrate its usage. Section 4 gives some details on the ideas behind the implementation. Since we are no longer in the commutative setting strings of variable length have to be used for storing terms instead of arrays of fixed size. Due to the realization of multiplication in the monoid and the computation of prefix Gröbner bases special data structures such as tries and ternary search trees supporting the reduction process are investigated. Section 5 outlines possible enhancements.

Acknowledgement The authors thank Christoph Kögl for valuable discussions on this paper.

2 Prefix Gröbner Bases in Monoid Rings

2.1 The General Case

For the field of rationals $\mathbb{Q}$ and a monoid $\mathcal{M}$ let $\mathbb{Q}[\mathcal{M}]$ denote the ring of all finite formal sums (called polynomials) $\sum_{i=1}^{n} \alpha_i \cdot w_i$ where $\alpha_i \in \mathbb{Q}\backslash\{0\}$ and $w_i \in \mathcal{M}$[1]. This ring is called the monoid ring over $\mathbb{Q}$ and $\mathcal{M}$. Since we want to do effective computations in $\mathbb{Q}[\mathcal{M}]$[2], we need an appropriate presentation of $\mathcal{M}$. In our context we always assume $\mathcal{M}$ to be presented by a string rewriting system consisting of a finite alphabet Σ and a finite set of rewriting rules $T \subset \Sigma^* \times \Sigma^*$, which is confluent and terminating with respect to some well-founded total admissible[3] ordering $\succeq$ on Σ^*. In particular, this means that the elements of $\mathcal{M}$ have unique representations by words in Σ^* and multiplication can be performed using the string rewriting system[4]. The empty word λ represents the unit in $\mathcal{M}$. Then $\succeq$ can be extended to $\mathbb{Q}[\mathcal{M}]$ and used to order the monomials occuring in a non-zero polynomial $f \in \mathbb{Q}[\mathcal{M}]$. This situation is similar to the ordinary polynomial ring. The largest monomial then is called the head monomial $\mathsf{HM}(f)$ and consists of the head term $\mathsf{HT}(f)$, and the head coefficient $\mathsf{HC}(f)$. For sets of polynomials F we denote $\mathsf{HT}(F) = \{\mathsf{HT}(f) \mid f \in F\}$. Notice that contrary to ordinary commutative polynomial rings we have the following phenomenon: for $f \in \mathbb{Q}[\mathcal{M}]$ and $w \in \mathcal{M}$ the equation $\mathsf{HT}(f * w) = \mathsf{HT}(f) \circ w$ need *not* hold (see e.g. [11]). This requires additional attention when characterizing Gröbner bases and implementing procedures to compute them as we will see later on.

As stated in the introduction, Gröbner bases are strongly related to reduction relations. For monoid and group rings various definitions of such relations are possible and can be found in the literature [15, 11, 10, 12]. Since the monoid elements can be seen as words or strings over the alphabet Σ it makes sense to speak of one element being a prefix of the other. For $u, v \in \mathcal{M}$ u is a prefix of v (as a string) if there exists $x \in \mathcal{M}$ such that u concatenated with x equals v. This will be expressed by writing $ux \equiv v$ where $\equiv$ represents identity of words. We define prefix reduction in this setting as follows: For two non-zero polynomials p, f in $\mathbb{Q}[\mathcal{M}]$, we say f **prefix reduces** p to q at a monomial $\alpha \cdot t$ of p in one step, denoted by $p \longrightarrow_f^{\mathrm{P}} q$, if $\mathsf{HT}(f)w \equiv t$ for some $w \in \mathcal{M}$ and

[1] We will use three different symbols for denoting multiplications: By $*$ we denote multiplication in the ring $\mathbb{Q}[\mathcal{M}]$, by $\circ$ multiplication in the monoid $\mathcal{M}$ and by $\cdot$ multiplication with scalars from $\mathbb{Q}$.

[2] All results given in this paper hold for arbitrary computable fields.

[3] i.e. compatible with concatenation and the empty word λ is the smallest element.

[4] The result of multiplying u and v is the normal form of the concatenated word uv.

$q = p - \alpha \cdot \mathsf{HC}(f)^{-1} \cdot f * w$. We will call a basis G of a (one or two-sided) ideal $\mathfrak{i}$ in $\mathbb{Q}[\mathcal{M}]$ a **prefix Gröbner basis** of $\mathfrak{i}$ if $\mathsf{HT}(\mathfrak{i}) = \{uw \mid u \in \mathsf{HT}(G), w \in \mathcal{M}\}$. Other characterizations will be given later on. G is called **prefix reduced** or **interreduced** if no polynomial in G is prefix reducible by another polynomial in G and all polynomials in G are monic.

However, before we give a completion procedure for prefix reduction, we have to take a closer look at what congruence prefix reduction using a set of polynomials $F \subset \mathbb{Q}[\mathcal{M}]$ describes. Since prefix reducing a polynomial corresponds to subtracting a *right* multiple of another polynomial and our monoid ring in general is not commutative, we can at most expect to describe a right ideal congruence. For a set $F \subseteq \mathbb{Q}[\mathcal{M}]$ let $\mathsf{ideal}_r(F)$ denote the right ideal generated by F. It turns out that while $\overset{*}{\longleftrightarrow}{}^{\mathrm{p}}_F \subseteq \equiv_{\mathsf{ideal}_r(F)}$ in general $\equiv_{\mathsf{ideal}_r(F)} \not\subseteq \overset{*}{\longleftrightarrow}{}^{\mathrm{p}}_F$, i.e. prefix reduction is not strong enough to describe the right ideal congruence. This is due to the above-mentioned fact that in general we cannot expect $\mathsf{HT}(f * w) = \mathsf{HT}(f) \circ w$ to hold for $f \in \mathbb{Q}[\mathcal{M}]$ and $w \in \mathcal{M}$. This defect can be repaired by introducing the concept of **prefix saturation**. A set $F \subseteq \mathbb{Q}[\mathcal{M}]$ is called prefix saturated if for each polynomial f in F and every element w in $\mathcal{M}$ we have that $f * w$ prefix reduces to zero in one step using a polynomial in F. A procedure for enumerating a prefix saturating set for a polynomial which terminates in case a finite such set exists is presented in [9]. We will refer to it as prefix.saturate. Now if F is prefix saturated we have $\overset{*}{\longleftrightarrow}{}^{\mathrm{p}}_F = \equiv_{\mathsf{ideal}_r(F)}$. Moreover, then a prefix Gröbner basis of the *right* ideal generated by F can be characterized similar to Buchberger's approach by prefix s-polynomials, based on the fact that prefix Gröbner bases of right ideals can be characterized as follows: A set $G \subseteq \mathbb{Q}[\mathcal{M}]$ is called a **prefix Gröbner basis** of $\mathsf{ideal}_r(G)$, if $\overset{*}{\longleftrightarrow}{}^{\mathrm{p}}_G = \equiv_{\mathsf{ideal}_r(G)}$, and $\longrightarrow^{\mathrm{p}}_G$ is confluent. The former condition can be achieved using prefix saturation and the latter can be tested by checking whether all prefix s-polynomials reduce to zero. Given two non-zero polynomials $p_1, p_2 \in \mathbb{Q}[\mathcal{M}]$, if there is $w \in \mathcal{M}$ such that $\mathsf{HT}(p_1) \equiv \mathsf{HT}(p_2)w$ the **prefix s-polynomial** is defined as $\mathsf{spol}(p_1, p_2) = \mathsf{HC}(p_1)^{-1} \cdot p_1 - \mathsf{HC}(p_2)^{-1} \cdot p_2 * w$.

Theorem 1 ([9]) *For a prefix saturated set $F \subseteq \mathbb{Q}[\mathcal{M}]$, the following statements are equivalent:*

(1) F is a prefix Gröbner basis of $\mathsf{ideal}_r(F)$.

(2) For all polynomials $f_1, f_2 \in F$ we have $\mathsf{spol}(f_1, f_2) \overset{*}{\longrightarrow}{}^{\mathrm{p}}_F 0$.

A procedure (referenced as prefix.groebner_basis_of_right_ideal_1 here) based on this theorem was presented in [9]. It terminates in case finite prefix saturated Gröbner bases exist. This is always the case e.g. for finite or free monoids and finite, free, and plain groups and with a small modification for context-free groups. In many cases specialized versions of our procedure are possible which make use of additional structural information on the monoid or group to speed up the computation (see e.g. 2.2 and 2.3 for special cases implemented so far).

However, there are more efficient ways to compute prefix Gröbner bases. Notice that computing the prefix s-polynomial for p_1 and p_2 where $\mathsf{HT}(p_1) \equiv \mathsf{HT}(p_2)w$ for some $w \in \mathcal{M}$ is directly related to prefix reducing $\mathsf{HM}(p_1)$ by p_2 and we get $p_1 \longrightarrow^{\mathrm{p}}_{p_2} p_1 - \mathsf{HC}(p_1) \cdot \mathsf{HC}(p_2)^{-1} \cdot p_2 * w = \mathsf{HC}(p_1) \cdot \mathsf{spol}(p_1, p_2)$. Hence, using the concept of **weakly prefix saturated** sets, i.e. for every $f \in F$ and every $w \in \mathcal{M}$ we have $f * w \overset{*}{\longrightarrow}{}^{\mathrm{p}}_F 0$, prefix Gröbner bases can also be characterized by interreduction:

Theorem 2 ([15, 11]) *A weakly prefix saturated set $F \subseteq \mathbb{Q}[\mathcal{M}]$ which is additionally prefix interreduced is a prefix Gröbner basis of $\mathsf{ideal}_r(F)$.*

This theorem gives rise to the following procedure:

prefix.groebner_basis_of_right_ideal_2

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, the reduced prefix Gröbner basis of $\mathsf{ideal}_r(F)$.
$G := \bigcup_{f \in F} \mathsf{prefix.saturate}(f)$;
while there is $g \in G$ such that $\mathsf{HT}(g)$ is prefix reducible by $G \backslash \{g\}$ **do**
  $G := G \backslash \{g\}$;
  $q := \mathsf{normal.form}(g, \longrightarrow_G^{\mathrm{p}})$; % Compute a monic normal form.
  if $q \neq 0$ **then** $G := G \cup \mathsf{prefix.saturate}(q)$;
endwhile

Similar to the case of solvable polynomial rings in [5], prefix Gröbner bases of *two-sided* ideals can be characterized by prefix Gröbner bases of right ideals which have additional properties.

Theorem 3 ([15, 11]) *For a set $G \subseteq \mathbb{Q}[\mathcal{M}]$ the following properties are equivalent:*

1. *G is a prefix Gröbner basis of $\mathsf{ideal}_r(G)$ and $\mathsf{ideal}_r(G) = \mathsf{ideal}(G)$.*
2. *G is a prefix Gröbner basis of $\mathsf{ideal}_r(G)$ and for all $a \in \Sigma$, $g \in G$ we have $a * g \in \mathsf{ideal}_r(G)$.*

This leads to the following procedure which need not terminate:

prefix.groebner_basis_of_two_sided_ideal

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, the reduced prefix Gröbner basis of $\mathsf{ideal}(F)$.
$H := \mathsf{prefix.groebner\_basis\_of\_right\_ideal\_2}(F)$;
$G := \{a * g \mid a \in \Sigma, g \in H\} \cup H$;
$G := \mathsf{prefix.groebner\_basis\_of\_right\_ideal\_2}(G)$;
while $G \neq H$ **do**
  $H := G$;
  $G := \{a * g \mid a \in \Sigma, g \in H\} \cup H$;
  $G := \mathsf{prefix.groebner\_basis\_of\_right\_ideal\_2}(G)$;
endwhile

Next we present special cases where the procedures always terminate and can be made more efficient by using additional structural knowledge on the monoid.

2.2 The Special Case of Free Monoid Rings

Free monoids allow simple presentations, namely of the form $(\Sigma, \emptyset)$ where Σ is the generating set and the set of rules is empty. Using such a presentation the procedures of the previous section can be simplified. As the set of rules is empty the polynomial itself is a prefix saturating set. Using this information procedure prefix.groebner_basis_of_right_ideal_2 can be specialized to procedure prefix.groebner_basis_of_right_ideal_fm and this in fact coincides with Mora's procedure for computing prefix Gröbner bases of right ideals in (free) non-commutative polynomial rings as presented in [14]. Similarly, prefix.groebner_basis_of_two_sided_ideal can be specialized to procedure prefix.groebner_basis_of_two_sided_ideal_fm by using prefix.groebner_basis_of_right_ideals_fm for computing the reduced prefix Gröbner bases of the right ideals.

2.3 The Special Case of Free Group Rings

In the case of free group rings the procedures of Section 2 can be replaced by procedures especially adapted to a special presentation of free groups: For a free group generated by Σ, the alphabet of

the presenting string rewriting system is $\Sigma \cup \Sigma^{-1}$ where Σ^{-1} is the set of the formal inverses of Σ and the set of rules is $T = \{(aa^{-1}, \lambda), (a^{-1}a, \lambda) \mid a \in \Sigma, a^{-1} \in \Sigma^{-1}\}$. Given such a presentation there exist saturatings set of the polynomial p either of the form $\{\lambda\}$ (if p consists of one monomial only) or consisting of two special polynomials $\{can(p), acan(p)\}$ (due to the fact that only two different terms in p can be brought to head position). See [15, 11] for more details.

prefix.saturate_fg

Given: A polynomial $f \in \mathbb{Q}[\mathcal{M}]$.
Find: $\{\lambda\}$ or $\{can(f), acan(f)\}$, a prefix saturating set for f.

if f contains only one monomial
  then $can(f) = acan(f) = \lambda$; **return**; **endif**
$ht := HT(f)$;
$acan(f) := f$;
while $ht = HT(acan(f))$ **do**
  $can(f) := acan(f)$;
  $\sigma :=$ last$(ht)^{-1}$ % last returns the last letter of a string
  $ht := ht * \sigma$;
  $acan(f) := acan(f) * \sigma$;
endwhile

Further the procedure for computing the reduced prefix Gröbner basis can be replaced by an adapted procedure which takes into account that mates $\{can(p), acan(p)\}$ can be replaced instead of individual polynomials p. This reduces the amount of necessary reduction steps as normal forms of $acan(p)$ are not computed if $can(p)$ already can be reduced. Moreover, on input F the Gröbner basis will not contain more than $2 \cdot |F|$ polynomials.

prefix.groebner_basis_of_right_ideal_fg

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, the reduced prefix Gröbner basis of ideal$_r(F)$.
$H := \bigcup_{f \in F}$ prefix.saturate_fg(f); (*)
$G := \emptyset$;
while $H \neq \emptyset$ **do**
  $H := H \backslash \{can(p), acan(p)\}$;
  if $|\ can(p)\ | = 1$ **OR** $|\ acan(p)\ | = 1$
    then $G := \{\lambda\}$; $H := \emptyset$;
    else $nf :=$ normal.form$(can(p), \longrightarrow^{p}_{G \cup H})$;
      if $|\ nf\ | = 1$
        then $G := \{\lambda\}$; $H := \emptyset$;
      elseif $nf < can(p)$
        then $G := G \cup$ prefix.saturate_fg(nf);
      else $nf :=$ normal.form$(acan(p), \longrightarrow^{p}_{G \cup H})$;
        if $|\ nf\ | = 1$
          then $G := \{\lambda\}$; $H := \emptyset$;
        elseif $nf < acan(p)$
          then $G := G \cup$ prefix.saturate_fg(nf);
        else $G := G \cup \{r, s\}$
        endif
      endif
  endif
  if $H = \emptyset$ **AND** a reduction occurred
    then $H := G$; $G := \emptyset$;
endwhile

The computation of prefix Gröbner bases for two-sided ideals can be simplified further. The original

procedure computes a prefix Gröbner basis, adds for each polynomial of this basis all left-multiples with the generators and then computes a prefix Gröbner basis of the new basis and so on. Since the new computation of the prefix Gröbner basis "forgets" that part of the input which has already been considered, many unnecessary steps are performed. To avoid this we implement some sort of preprocessing when computing the left extensions of our bases: the new elements obtained by left-multiplication with the generators are first reduced to normal form and then prefix saturated. We present an adaption of this idea for the case of free group rings:

extend.left_fg

Given: A finite set $H \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, a preprocessed version of $\{a * g \mid a \in \Sigma, g \in H\} \cup H$.
$G := H$;
for all $a \in \Sigma$ **do**
  for all $h \in H$ **do**
    $r := a * h$;
    $nf :=$ normal.form$(r, \longrightarrow_G^{\mathrm{p}})$;
    if $nf \neq 0$ **then** $G := G \cup \{can(nf), acan(nf)\}$;
  endfor
endfor

The procedure for prefix Gröbner bases of two-sided ideals in free group rings uses prefix.modified_groebner_basis_of_right_ideal_fg which is essentially procedure prefix.groebner_basis_of_right_ideal_fg omitting the line marked with (*).

prefix.groebner_basis_of_two_sided_ideal_fg

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, the *reduced* prefix Gröbner basis of ideal(F).
$H := \bigcup_{f \in F}$ prefix.saturate_fg(f);
$H :=$ prefix.modified_groebner_basis_of_right_ideal_fg(H);
$G :=$ extend.left_fg(H);
$G :=$ modified.prefix.groebner_basis_of_right_ideal_fg(G);
while $G \neq H$ **do**
  $H := G$;
  $G :=$ extend.left_fg(H);
  $G :=$ prefix.modified_groebner_basis_of_right_ideal_fg(G);
endwhile

3 MRC

MRC the **M**onoid **R**ing **C**ompletion program implements the procedures described in the previous sections. The user can choose between the following procedures:

- prefix.reduce_set to prefix interreduce a set of polynomials,
- prefix.groebner_basis_of_right_ideal_1 as mentioned in Section 2.1,
- prefix.groebner_basis_of_right_ideal_2 as described in Section 2.1,
- prefix.groebner_basis_of_right_ideal_fm as mentioned in Section 2.2,
- prefix.groebner_basis_of_right_ideal_fg as described in Section 2.3,
- prefix.groebner_basis_of_two_sided_ideal as described in Section 2.1,
- prefix.groebner_basis_of_two_sided_ideal_fm as mentioned in Section 2.2,
- prefix.groebner_basis_of_two_sided_ideal_fg as described in Section 2.3.

The input for the procedures computing the interreduced set of a set of polynomials and prefix Gröbner bases of right ideals is a monoid presentation and a set of polynomials. The monoid

presentation is given by a set of generators, an ordering, and a set of rules forming a convergent string rewriting system which can in many cases be obtained using the Knuth-Bendix completion procedure (e.g. the system COSY developed at Kaiserslautern or the system KBMAG developed at Warwick). Its output can then be transformed into an input for MRC. The result produced by MRC is a set of polynomials representing the respective prefix Gröbner basis.

The procedures for prefix Gröbner bases of two-sided ideals, too, take a monoid presentation and a set of polynomials as input. They produce as output a sequence of polynomial sets, each representing the prefix Gröbner basis of the right ideal computed after having left extended the last one. On termination the prefix Gröbner basis of the two-sided ideal generated from the original set of polynomials is displayed. The procedure can be run interactively, that is the user can decide to continue or to interrupt the computation after each newly computed prefix Gröbner basis, or without user interaction in batch mode.

Statistical information concerning memory consumption and run time are provided, too. They allow assessments on the efficency of the procedures implemented.

In [13] connections between the word problem for monoids and groups and the ideal membership problem in free monoid and free group rings, respectively, as well as connections between the submonoid problem and the subalgebra problem and between the subgroup problem and the one-sided ideal membership problem are proven. Hence prefix Gröbner bases can be used to study group theoretical problems. Instead of providing the tedious syntax of the input of MRC (see [17] for the user reference) we want to illustrate its usage by giving an example of applying prefix Gröbner bases:

For a group $\mathcal{G}$ and a finite set of generators $u_1, \ldots, u_k$ of a subgroup $\mathcal{H}$ the generalized word problem is to decide whether $g \in \mathcal{G}$ is also an element of $\mathcal{H}$. This problem can be linked to the right ideal membership problem for the group ring $\mathbb{Q}[\mathcal{G}]$ as follows: A group element g is in $\mathcal{H}$ if and only if the polynomial $g - 1 \in \mathbb{Q}[\mathcal{G}]$ is in the right ideal generated by the polynomials $u_1 - 1$, $\ldots$, $u_k - 1$ in $\mathbb{Q}[\mathcal{G}]$. Hence, this question can be attacked using prefix Gröbner bases as well. See [9, 13] for further theoretical background on this topic.

Let the group $\mathcal{G}$ be presented by $(a; aaaaaa = aA = Aa = 1)$ and the subgroup $\mathcal{H}$ of $\mathcal{G}$ generated by aaa. A convergent string rewriting system presenting $\mathcal{G}$ can be computed from the relations using COSY: $\Sigma = \{a, A\}$, where A denotes the formal inverse of a, $T = \{(aA, \lambda), (Aa, \lambda), (aaa, AA), (AAA, aa)\}$ and the length lexicographical ordering in induced by $A \succ a$ (the precedence is described by numbering the letter in the input for MRC below). The polynomial set is $\{aaa - 1\}$. MRC uses the string `$\lambda$` to encode the empty word λ.

```
You have selected the following parameters:

Group:
Alphabet:
( A 2 ) ( a 1 );
Ordering:
length-lexicographic;
Rules:
( aA  $\lambda$ )
( Aa  $\lambda$ )
( aaa AA )
( AAA aa );

Generating set of polynomials:
( 1 / 1 * aaa + -1 / 1 * $\lambda$ ) ;
Method:  prefix.groebner_basis_of_right_ideal_1

Reduced prefix Groebner basis:
```

```
(1/1 * AA + -1/1 * a)
(1/1 * aa + -1/1 * A)
```

Now it is easy to decide whether a given element of $\mathcal{G}$ lies in $\mathcal{H}$ or not: $AAA \in \mathcal{H}$ since $AAA - 1 \longrightarrow^{\mathrm{P}}_{AA-a} 0$ but $AA \notin \mathcal{H}$ as $AA - 1 \longrightarrow^{\mathrm{P}}_{AA-a} a - 1$ and the latter is irreducible.

4 Implementation

4.1 Goals and General Structure

MRC was designed for two main purposes: to compute prefix Gröbner bases in monoid and group rings as well as to determine the feasibility of the procedures.

One goal was to get a simple, extendable, and fast implementation. C++ was choosen as implementation language because it is widely available, provides concepts from object-orientation like inheritance, facilitates reuse of code (for example through template classes), and leads to reasonably fast code if certain coding principles are followed. Further there exist some implementations of well known data structures like the LEDA-Library (see [1]). Thus MRC is implemented in C++ using the LEDA-Library on SUN SparcWorkstations under Solaris 2.5. A first implementation of prefix reduction for monoid and group rings and the respective completion procedures took about three weeks from the first conceptual model to the first running version. One more week was spent on extending the program for two-sided ideals, fixing bugs and memory leaks.

Based on the procedures in [11] the following approach was taken: undertake an object oriented analysis and design, write the procedures top down, and test the modules bottom up. The object oriented design resulted in the top level class diagram depicted in Figure 1. Only the module names and the data sections are shown, the methods have been ommitted. Additional class diagrams not shown here exist for the concepts "monoid" and "set of polynomials".

The abstract procedures of [11] were implemented using well known data structures provided by the LEDA-Library. Notable exceptions are the data structures facilitating efficient reduction, or more precisely pattern matching on strings needed for the reduction process. In Section 4.2 we explain why special data structures are needed in comparison to Gröbner bases for commutative structures, before we describe these data structures in Section 4.3.

4.2 Differences to Commutative Polynomial Rings

Readers not familiar with string rewriting techniques might wonder what differences there are between this approach to Gröbner bases for non-commutative structures and the approaches known for Gröbner bases in commutative structures.

Polynomials are usually represented as ordered lists of monomials and the main operations involved in the reduction process and the computation of s-polynomials are multiplication, comparison, matching, and unification of terms. In the commutative case terms can be represented by the exponents of the generators and stored in arrays of a fixed length, namely the number of generators. The main operations then translate to operations on these arrays and can be done efficiently. In the non-commutative setting representations of terms have to include the sequence of the letters, hence strings of variable length have to be stored. All operations now have to be performed on strings and especially multiplication, which is performed by concatenating the two terms and computing their normal form, can be costly depending on the presentation of the monoid. Matching and unification in the case of prefix reduction involve finding prefixes of strings, but will become more complicated when implementing other reduction relations.

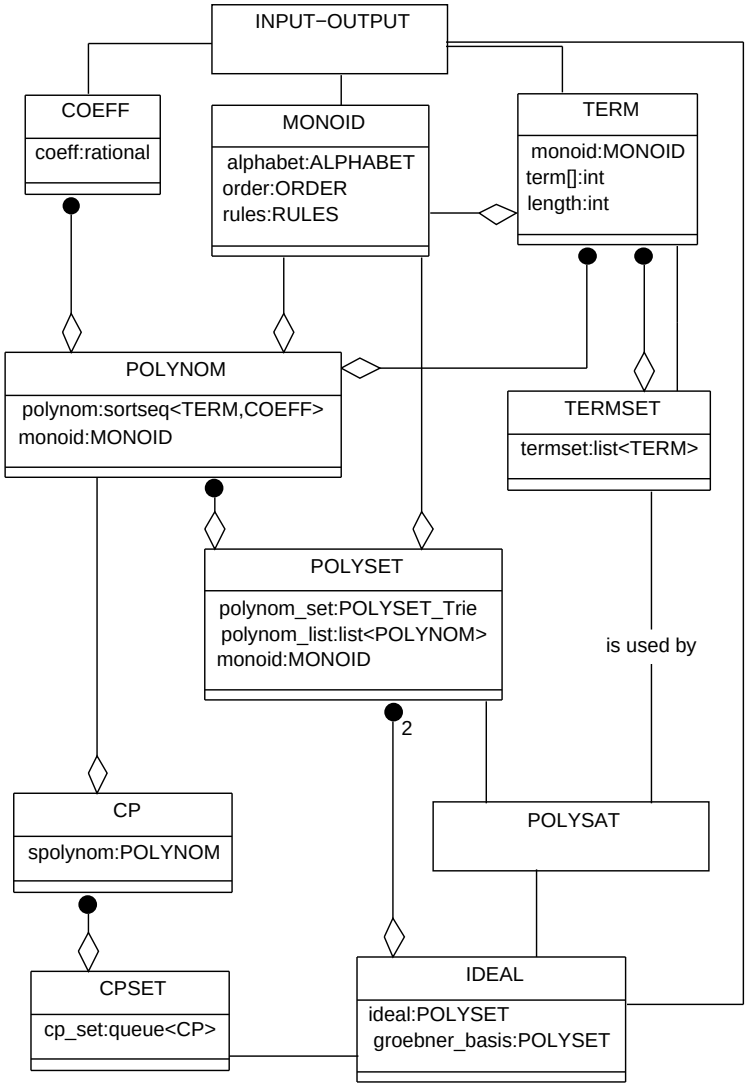

Figure 1: System Structure

Another difference stems from the fact that the ordering on the monoid in general is no longer compatible with multiplication. In the commutative case multiplication of a polynomial with a term can be done by multiplying one monomial after the other without changing the ordering of the monomials in the new polynomial. In the non-commutative case this becomes more complicated: after multiplying each monomial we have to combine those monomials which now have the same term and reorder all occurring monomials in the new polynomial. This of course has influence on the polynomial operations such as multiplication or reduction. Moreover, as $\mathsf{HT}(f * w) \neq \mathsf{HT}(f) \circ w$ is possible, polynomials have to be saturated (see Section 2.1) which adds additional complexity.

Since prefix matching of strings turned out to be the crucial operation when computing prefix Gröbner bases in monoid and group rings, special attention was paid to speeding up this process. Some observations are given in the next section.

4.3 Data Structures for Prefix Reduction

The elements of the monoid are called strings, words or terms. Words can be seen as sequences of characters. In order to reduce one word with another word it is necessary to find matches. In other words: the normalization procedure for terms tries to determine whether a left hand side of a rule of the monoid is a substring of the term to be normalized. This is equivalent to the question whether a left hand side of a rule is a prefix of any suffix of the term to be normalized. E.g. ba is prefix of a suffix of $abab$, namely bab. Thus a rule with ba as its left hand side can be applied to reduce $abab$ at the starting position of its suffix bab. Reduction of polynomials as defined in Section 2 is based on prefix reduction of terms. So one important operation is to find all the terms in a set of terms (e.g. the head terms of a set of polynomials) which are prefixes of a given term (which can in fact be a suffix of another term). This is supported by prefix trees. Prefix trees are trees in which common prefixes of the words stored in the tree are shared.

A prefix tree represents a set of keys (in this case terms) which are used to find the contents associated to each key. The (key, content)-pairs can be, for example, (left hand side of a rule r, rule r) or (head term of a polynomial p, polynomial p). Note, that the head terms of two or more polynomials might be equal and thus there might be more than one polynomial associated with the same key.

Two kinds of prefix trees were implemented: tries, a fairly standard data structure, and ternary search trees, a recently rediscovered variation of tries (see [3]). Both are described in the next subsections followed by a comparison between the two data structures.

4.3.1 Trie

The following definition is based on the description oin[8], see there for more details and examples. Note that there are several slightly different definitions of tries in the literature.

Let k be the number of characters of the underlying alphabet. A trie is a k-ary tree. Each node of the tree consists of k pointers to the subtrees at the next level, one for each character, and one pointer to the content (which is empty if no key corresponding to the current node exists). The content associated with a key is retrieved as follows:

- The first character of the key becomes the current character. The root node the current node.
- The current character is used as index into the array of the current node and the pointer to the subtree is followed. Note that it is usually assumed that this index operation takes constant time. That node becomes the current node, the next character the current character.
- If no subtree exists, then the key is not stored in the tree.
- This is repeated until all characters of the key were found unless the key does not exist.
- The content of the last node obtained is the content associated with the key. If it is empty, then no such key is stored in the tree.

If a key has m characters the search path ends at level m of the tree. The keys are not stored explicitly. The presence or absence of contents implies that a key ends or does not end at this node, respectively.

4.3.2 Ternary Search Tree

Ternary search trees as implemented in MRC were presented in [3], see there for more details and examples. In contrast to tries as described in the previous section alphabets of variable length pose no problems.

The characters are ordered by a total ordering which has the properties as defined in Section 2. A node of the ternary search tree consists of six elements: the character represented by the node, a pointer to the subtree of characters which are smaller with respect to the ordering, a pointer to the subtree of characters which are larger with respect to the ordering, a pointer to the subtree for the next level, and a pointer to the content. The character and the pointers to the subtrees of smaller and larger characters form a binary search tree which replaces the array of the trie. In order to explain the structure of a ternary search tree (tst) more clearly the insertion procedure is described (deletion and lookup of keys follow the same scheme):

- If a term is to be inserted, its first character is taken as the current character and the root node of the tst as the current node.
- If the current character is smaller than the character at the current node the pointer to the subtree of the smaller characters is followed and that node becomes the current node.
- If the current character is larger than the character at the current node the pointer to the subtree of the larger characters is followed and that node becomes the current node.
- If the respective node does not exist, it is created storing the current character in the new node, and the new node becomes the current node.
- If the current character is equal to the character at the current node the pointer to the subtree of the next level is followed. That node becomes the current node, and the next character the current character.
- If no subtree node exists, it is created, storing the next character of the term in the new node. The new node becomes the current node, the next character the current character.
- This is repeated until all characters of the term were found or nodes for them created.
- The last node is assigned the content (rule or polynomial) associated with the term inserted.

4.3.3 Comparison of Tries and Ternary Search Trees

Tries and ternary search trees are both suited as data structures for prefix trees.

The major advantage of tries is that insertion, lookup, and deletion of a term have complexity $O(m)$ where m is the length of the term while ternary search trees have an average complexity for these operations of $O(log(k) \cdot m)$ where k is the number of characters of the alphabet and m the length of the term.

The major disadvantage of tries is that for each node of the tree (except for the leaves) memory for $k+1$ pointers has to be allocated where k is the size of the alphabet. The latter also slows down the creation of nodes because each of the pointers has to be initialized. Ternary search trees need to store only 4 pointers and the character in each node, but there exist more nodes. Another disadvantage of tries is that changing the alphabet requires a total reorganization of the trie structure.

Practical tests showed the following phenomenom. There are examples (see [17]) where no difference of the runtime was observed for the computation of the Gröbner base, but where the ternary search tree used much less space during the computation than the trie. Only for cases of small alphabets or almost complete trees, where each sequence of characters up to a certain length is stored tries might perform better than ternary search trees.

Besides these advantages both data structures do not allow storing the order in which elements were inserted. For implementing a fair strategy for the computation of Gröbner bases additional data structures are used.

When enumerating all elements the trie in general will involve more complexity than the ternary search trees where in general less pointers are involved.

5 Enhancements

Having implemented a core of procedures several enhancements are possible now. As for the special cases of free monoids and groups, the set of procedures can be extended to cover other special structures such as plain and context-free groups. Further monoid rings over reduction rings can be implemented. Moreover, procedures for other specialized reduction concepts to treat abelian monoid rings and polycyclic group rings exist which can be added to the system.

On the other hand there is a potential for making enhancements concerning the time and space consumption of the existing procedures. The ternary search tree is one such enhancement already realized. Other possibilities are the use of a finite state automata for representing the set of rules to speed up the pattern matching process needed for the computation of normal forms. It is planned to integrate MRC into the system XSSR currently developed at the Gesamthochschule Kassel and the University of Kaiserlautern in order to use the string rewriting facilities to perform all operations involving the monoid (i.e. completion of the presentation, computation of the multiplication, matching and unification of terms for performing reduction saturation and s-polynomial computation). Moreover, as XSSR is intended to assist people working in monoid and group theory, the specialized Gröbner basis procedures are of interest in this setting as well.

Another important task at hand is to determine the time and space complexity of the procedures presented and their influence on enhancements of the implementation. Right now this is done for the case of free monoid and free group rings.

References

[1] LEDA: http://www.mpi-sb.mpg.de/leda/leda.html.

[2] J. Apel, W. Lassner. An extension of Buchberger's algorithm and calculations in enveloping fields of Lie algebras. *JSC*, 6, 1988.

[3] J. Bently, R. Sedgewick. Fast algorithms for sorting and searching strings. In *8th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[4] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal.* PhD thesis, Universität Innsbruck, 1965.

[5] A. Kandri-Rody, V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. *JSC*, 9, 1990.

[6] B. J. Keller. Alternatives in implementing noncommutative Gröbner basis systems. In *Proc. SRT'95, Monte Verita.* Birkhäuser, 1998.

[7] W. Lassner. Symbol representations of noncommutative algebras. In *EUROCAL'85*, 1985.

[8] H. Lewis, L. Denenberg. *Data Structures & Their Algorithms.* Harper Collins, 1991.

[9] K. Madlener, B. Reinert. Computing Gröbner bases in monoid and group rings. *Proc. ISSAC'93*, 1993.

[10] K. Madlener, B. Reinert. A generalization of Gröbner bases algorithms to nilpotent group rings. *AAECC*, 8(2), 1997.

[11] K. Madlener, B. Reinert. String rewriting and Gröbner bases – a general approach to monoid and group rings. In *Proc. SRT'95, Monte Verita*, Birkhäuser, 1998.

[12] K. Madlener, B. Reinert. A generalization of Gröbner basis algorithms to polycyclic group rings. *JSC*, 1998.

[13] K. Madlener, B. Reinert. Relating rewriting techniques on monoids and rings: Congruences on monoids and ideals in monoid rings. *TCS*, 1998.

[14] F. Mora. Gröbner bases for non-commutative polynomial rings. In *Proc. AAECC-3*, 1985.

[15] B. Reinert. *On Gröbner Bases in Monoid and Group Rings.* PhD thesis, Universität Kaiserslautern, 1995.

[16] B. Reinert, T. Mora, K. Madlener. A note on coset enumeration. submitted to *ISSAC'98*, 1998.

[17] B. Reinert, D. Zeckzer. User reference for MRC: A system for computing Gröbner bases in monoid and group rings. Technical report, Universität Kaiserslautern, 1998.