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

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

Abstract

Gröbner bases and Buchberger's algorithm have been generalized to monoid and group rings. In this paper we summarize procedures from this field and present a description of their implementation in the system MRC V 1.0.

Keywords
Monoid and group rings, Gröbner bases, prefix reduction

*MRC stands for **M**onoid **R**ing **C**ompletion.

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

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 [8] 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 [16, 17] 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 [7].

The next step of generalizing Gröbner bases was to study arbitrary monoid and group rings. The theoretical background on prefix Gröbner bases for monoid and group rings was first presented at ISSAC'93 by Madlener and Reinert in [10]. Different specialized definitions of reduction relations followed [19, 14, 11, 12]. Applications of these methods can be found in [15] where connections are presented

- between the word problem for monoids and the ideal membership problem for free monoid rings;
- between the word problem for groups and the ideal membership problem for free group rings;
- between the submonoid problem and the subalgebra problem for monoid rings;
- and between the subgroup problem and the one-sided ideal membership problem for group rings.

Another link to group theory is provided in [20]: the well-known procedure of Todd and Coxeter for enumerating cosets of a subgroup can be rephrased using prefix Gröbner bases (in fact the implementation used to compute the examples there was done in MRC V 1.0).

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 V 1.0 and provides examples 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 for supporting the reduction process are investigated. Section 5 outlines possible enhancements.

In Appendix A the proofs of some new results presented in Section 2 are given. In Appendix B additional information about the classes is given. It is intended to supplement the comments of the source code.

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

2 Prefix Gröbner Bases in Monoid and Group Rings

In the following sections we summarize results from the theory of prefix Gröbner bases in monoid and group rings: First of all we develop the general case (see Section 2.1) similar to [19] and [14]. Procedures for prefix saturation and Gröbner basis computation are presented. Then we show how these procedures can be modified to enumerate a prefix Gröbner basis (see Section 2.2) if e.g. the prefix saturating process does not terminate. In Sections 2.3 to 2.6 special saturation procedures are presented using special presentations for free monoids and free, plain and context-free groups. For all these cases finite prefix Gröbner bases can be computed for finitely generated right ideals. Finally we present a procedure which simulates the Todd-Coxeter enumeration of cosets (see Section 2.7).

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) $\sum_{i=1}^{n} \alpha_i \cdot w_i$ with coefficients $\alpha_i \in \mathbb{Q}\backslash\{0\}$ and terms $w_i \in \mathcal{M}$[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]. Additionally, we require T to be interreduced. 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 occurring 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. [14]). 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 [19, 14, 11, 12]. Due to our presentation of the monoid by

[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.

a convergent string rewriting system, the monoid elements will be identified with the unique normal form of the strings representing them. Hence 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:

Definition 1 *Let p, f be two non-zero polynomials 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 $p \longrightarrow^{\mathrm{p}}_f q$, if*

(a) $\mathsf{HT}(f)w \equiv t$ *for some* $w \in \mathcal{M}$, *and*

(b) $q = p - \alpha \cdot \mathsf{HC}(f)^{-1} \cdot f * w$.

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 $\mathsf{HT}(\mathfrak{i}) = \{uw \mid u \in \mathsf{HT}(G), w \in \mathcal{M}, uw$ in normal form with respect to the set of rules T defining our monoid $\mathcal{M}\}$. 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 all $f \in \mathbb{Q}[\mathcal{M}]$ 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. Procedure 1 (see page 5) for enumerating a prefix saturating set for a polynomial which terminates in case a finite such set exists was presented in [10].

This process in general will not terminate. Then additional effort has to be exercised in order to give an enumeration process for a prefix Gröbner basis in the completion procedures below (see Section 2.2). However, for many structures finite prefix saturating sets exist and specialized prefix saturation procedures can be implemented (always with respect to special presentations of the monoids and groups). We will outline the cases of free monoids and free, plain, and context-free groups in Section 2.3 - Section 2.6.

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

prefix.saturate

Given: A polynomial $f \in \mathbb{Q}[\mathcal{M}]$.
Find: A (possibly infinite) prefix saturating set S for f.

$S := \{f\}$;
$H := \{f\}$;
while $H \neq \emptyset$ **do**
  $q :=$ remove(H);
  % Remove an element using a fair strategy, i.e., no element is left in H for ever
  $t := \mathsf{HT}(q)$;
  for all $w \in C(t) = \{w \in \Sigma^* \mid tw \equiv t_1 t_2 w \equiv t_1 l, t_2 \neq \lambda \text{ for some } (l, r) \in T\}$ **do**
    % $C(t)$ contains special overlaps between t and left hand sides of rules in T
    $q' := q * w$;
    if $q' \not\longrightarrow^{\mathrm{p}}_S 0$ and $q' \neq 0$
      then $S := S \cup \{q'\}$;
        $H := H \cup \{q'\}$;
    endif
  endfor
endwhile

Procedure 1: Prefix saturation

can be obtained 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 2 ([10]) *For a prefix saturated set F of polynomials in $\mathbb{Q}[\mathcal{M}]$, 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 $\mathsf{spol}(f_1, f_2) \xrightarrow{*}{}^{\mathrm{p}}_F 0$.

Procedure 2 (see page 6) based on this theorem was presented in [10].

It terminates for finite or free monoids and finite, free, and plain groups and with a small modification for context-free groups (if appropriate presentations of the monoids and groups are used).

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 p_1 by p_2 and we get $p_1 \longrightarrow^{\mathrm{p}}_{p_2} p_1 - \mathsf{HC}(p_1) \cdot \mathsf{HC}(p_2)^{-1} \cdot p_2 * w = \mathsf{HC}(p_1) \cdot \mathsf{spol}(p_1, p_2)$. Using the concept of **weakly prefix saturated** sets, i.e. for every $f \in F$ and every $w \in \mathcal{M}$ we have $f * w \xrightarrow{*}{}^{\mathrm{p}}_F 0$, prefix Gröbner bases can also be characterized by interreduction:

Theorem 3 ([19, 14]) *A weakly prefix saturated set $F \subseteq \mathbb{Q}[\mathcal{M}]$ which is additionally prefix interreduced is a prefix Gröbner basis of $\mathsf{ideal}_r(F)$.*

prefix.groebner_basis_of_right_ideal_1

Given: A finite set of polynomials $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, a prefix Gröbner basis of $\mathsf{ideal}_r(F)$.

$G := \bigcup_{f \in F} \mathsf{prefix.saturate}(f)$;
% G is prefix saturated
$B := \{(q_1, q_2) \mid q_1, q_2 \in G, q_1 \neq q_2\}$;
while $B \neq \emptyset$ **do**
  % Test if statement 2 of theorem 2 is valid
  $(q_1, q_2) := \mathsf{remove}(B)$;
  % Remove an element using a fair strategy
  if $\mathsf{spol}_p(q_1, q_2)$ exists
    % The s-polynomial is not trivial
    then h:= $\mathsf{normal.form}(\mathsf{spol}_p(q_1, q_2), \longrightarrow_G^{\mathrm{P}})$;
      % Compute a normal form using prefix reduction
      if $h \neq 0$
        then $G := G \cup \mathsf{prefix.saturate}(h)$;
          % G is prefix saturated
          $B := B \cup \{(f, \tilde{h}), (\tilde{h}, f) \mid f \in G, \tilde{h} \in \mathsf{prefix.saturate}(h)\}$;
      endif
  endif
endwhile

Procedure 2: Prefix Gröbner basis computation for right ideals

prefix.interreduce

Given: A finite set of polynomials $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: The finite interreduced set of polynomials G of F.
$G := F$;
while there is $g \in G$ which 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 $G := G \cup \{q\}$;
  endif
endwhile

Procedure 3: Prefix interreduction

This theorem is the basis for a more efficient implementation of the previous procedure using interreduction instead of computing prefix s-polynomials. We need Procedure 3 for computing the interreduced set of polynomials of a given set of polynomials and we need Procedure 4 (see page 7) for checking whether an interreduced set of polynomials is additionally weakly prefix saturated.

weakly.prefix.saturated_check

Given: A finite interreduced set of polynomials $G \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: $H = \emptyset$ if and only if G is weakly prefix saturated.

$H := \emptyset$;
for all $g \in G$ **do**
  $S :=$ prefix.saturate(g);
  while $S \neq \emptyset$ **do**
    $s :=$ remove(S);
    $nf :=$ normal.form$(s, \longrightarrow_G^{\mathrm{p}})$;
    if $nf \neq 0$
      then $H := H \cup \{nf\}$;
    endif
  endwhile
endfor

Procedure 4: Checking for weakly prefix saturatedness

prefix.groebner_basis_of_right_ideal_2

Given: A finite set of polynomials $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: G, a reduced prefix Gröbner basis of ideal$_r(F)$.

$G :=$ prefix.interreduce(F);
$H :=$ weakly.prefix.saturated_check(G);
while $H \neq \emptyset$ **do**
  $G :=$ prefix.interreduce$(G \cup H)$;
  $H :=$ weakly.prefix.saturated_check(G);
endwhile

Procedure 5: Interreduced prefix Gröbner basis computation for right ideals

Theorem 4 *Let H be the set generated by procedure* weakly.prefix.saturated_check. *Then $H = \emptyset$ if and only if G is weakly prefix saturated.*

Proof: See Appendix A.

If the set H is not empty, then at least the polynomials in H have to be added in order to fulfill the property of being weakly prefix saturated. This of course can violate the precondition that the set is prefix reduced. Hence the two processes of prefix interreducing the set G and adding the set H resulting from checking G for weakly prefix saturatedness have to be intertwined leading to Procedure 5 .

Theorem 5 *Let G be the set generated by procedure* prefix.groebner_basis_of_right_ideal_2 *on a finite input $F \subseteq \mathbb{K}[\mathcal{M}]$. Then G is a reduced prefix Gröbner basis.*

Proof: See Appendix A.

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 $\mathsf{ideal}(F)$.

$H :=$ prefix.groebner_basis_of_right_ideal_2(F);
$G := \{a * g \mid a \in \Sigma, g \in H\} \cup H$;
$G :=$ prefix.groebner_basis_of_right_ideal_2(G);
while $G \neq H$ **do**
  $H := G$;
  $G := \{a * g \mid a \in \Sigma, g \in H\} \cup H$;
  $G :=$ prefix.groebner_basis_of_right_ideal_2(G);
endwhile

Procedure 6: Prefix Gröbner basis computation for two-sided ideals

Theorem 6 *Let $F \subseteq \mathbb{K}[\mathcal{M}]$ be a finite set of polynomials. In case $\mathsf{ideal}_r(F)$ has a finite reduced prefix Gröbner basis, the procedure* prefix.groebner_basis_of_right_ideal_2 *terminates.*

Proof: See Appendix A.

Thus, procedure prefix.groebner_basis_of_right_ideal_2 terminates for finite or free monoids and finite, free, and plain groups and with a small modification for context-free groups (if appropriate presentations of the monoids and groups are used).

In the Sections 2.3 - 2.6 specialized versions of the prefix Gröbner basis procedures are presented which make use of additional structural information on the monoid or group to speed up the computation.

We close this section by showing how similar to the case of solvable polynomial rings in [6], prefix Gröbner bases of *two-sided* ideals can be characterized by prefix Gröbner bases of right ideals which have additional properties.

Theorem 7 ([19, 14]) *For a set of polynomials $G \subseteq \mathbb{Q}[\mathcal{M}]$ 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 Procedure 6 which may not terminate.

2.2 An Algorithm for Enumerating a Prefix Gröbner Basis

As already mentioned in the previous section, the procedure prefix.groebner_basis_of_right_ideal_1 does not terminate in general. In the case of

prefix.saturate_enum_init

Given: A polynomial $f \in \mathbb{Q}[\mathcal{M}]$.
Find: Initialize S, the saturating set and H, the testing set.

$S := \{f\}$;
$H := \{f\}$;

Procedure 7: Initialization of the prefix saturation enumeration

prefix.saturate_enum_step

Given: S and H
Find: A set R representing the next polynomials of the saturating set for f generated by overlapping the first element of H with the rules.

$q :=$ remove(H);
% Remove an element using a fair strategy, i.e., no element is left in H for ever
$t :=$ HT(q);
for all $w \in C(t) = \{w \in \Sigma^* \mid tw \equiv t_1 t_2 w \equiv t_1 l, t_2 \neq \lambda \text{ for some } (l, r) \in T\}$ **do**
  % $C(t)$ contains special overlaps between t and left hand sides of rules in T
  $q' := q * w$;
  if $q' \not\to_S^{\mathrm{P}} 0$ and $q' \neq 0$
    then $S := S \cup \{q'\}$;
      $H := H \cup \{q'\}$;
      $R := R \cup \{q'\}$;
  endif
endfor

Procedure 8: Making one step of the prefix saturation enumeration

non termination it is nevertheless interesting to give a semi-decision procedure for the ideal membership problem: does a given polynomial lie in the ideal generated by a set of polynomials. In order to decide this, a Gröbner basis of the set generating the ideal is enumerated and it is tested whether the given polynomial can be reduced to zero which is equivalent to its belonging to the ideal. In case polynomial is reduced to zero with respect to the Gröbner basis enumerated so far, the procedure terminates with a positive answer. If the polynomial does not belong to the ideal, the answer is no if the enumeration process stops, else no answer is given.

The algorithms prefix.saturate and prefix.groebner_basis_of_right_ideal_1 have to be modified accordingly. First the algorithm for prefix saturation is split into two parts: initialization and performing one saturation step leading to the two Procedures 7 and 8. The while-loop is integrated in the enumeration procedure. The same is true for the Gröbner basis computation itself. It is split into two parts: initialization and handling of one s-polynomial. The while-loop is integrated in the enumeration procedure.

These procedures together with the sets H and S form a so called saturator. Instead of computing the saturating set a saturator is initialized for each of the polynomials of the input set. Then one step of all saturators is taken and the resulting polynomials as well as the new s-polynomials are added to the Gröbner basis and the set of s-

enumerate_prefix_groebner_basis_of_right_ideal

Given: A finite set of polynomials $F \subseteq \mathbb{Q}[\mathcal{M}]$.
Find: Enumerate a prefix Gröbner basis of $\mathsf{ideal}_r(F)$.

$SAT := \bigcup_{f \in F} \mathsf{prefix.saturate\_enum\_init}(f)$;
% initialize the saturators
$G := F$;
$B := \{(q_1, q_2) \mid q_1, q_2 \in G, q_1 \neq q_2\}$;
$S := \bigcup_{sat \in SAT} \mathsf{prefix.saturate\_enum\_step}(sat)$;
% compute the polynomials of the first step of all saturators
$G = G \cup S$;
$B := B \cup \{(g, s), (s, g) \mid g \in G, s \in S\}$;
while $B \neq \emptyset$ **do**
    % Test if statement 2 of theorem 2 is valid
    $(q_1, q_2) := \mathsf{remove}(B)$;
    % Remove an element using a fair strategy
    if $\mathsf{spol}_p(q_1, q_2)$ exists
        % The s-polynomial is not trivial
        then h:= $\mathsf{normal.form}(\mathsf{spol}_p(q_1, q_2), \longrightarrow^{\mathrm{p}}_G)$;
            % Compute a normal form using prefix reduction
            if $h \neq 0$
                then $G := G \cup \{h\}$;
                    $SAT := SAT \cup \mathsf{prefix.saturate\_enum\_init}(h)$;
                    % initialize a new saturator
            endif
    endif
    $S := \bigcup_{sat \in SAT} \mathsf{prefix.saturate\_enum\_step}(sat)$;
    $G = G \cup S$;
    $B := B \cup \{(g, s), (s, g) \mid g \in G, s \in S\}$;
endwhile

Procedure 9: Enumeration of a prefix Gröbner basis of a right ideal

polynomials, respectively. A saturator terminates if the set H becomes empty. It will then be removed from the set of saturators SAT. This leads to Procedure 9.

2.3 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 Section 2.1 can be simplified. As the set of rules is empty the polynomial itself is a prefix saturating set. Using this information the procedures prefix.groebner_basis_of_right_ideal_1 and weakly.prefix.saturated_check can be specialized by substituting $\{p\}$ for $\mathsf{prefix.saturate}(p)$. On the other hand, as prefix s-polynomials are strongly related to prefix reduction at head terms, prefix Gröbner bases can be com-

prefix.saturate_fg

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

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

Procedure 10: Prefix saturation in free group rings

puted using Mora's procedure for computing prefix Gröbner bases of right ideals in (free) non-commutative polynomial rings as presented in [16]. This procedure is equivalent to the procedure prefix.interreduce presented in Section 2.1 with F as the input set and G being the reduced Gröbner Basis of $\mathsf{ideal}_r(F)$.

Further, prefix.groebner_basis_of_two_sided_ideal can be specialized to procedure prefix.groebner_basis_of_two_sided_ideal_fm by using prefix.interreduce for computing the reduced prefix Gröbner bases of the right ideals. However, this procedure may not terminate.

2.4 The Special Case of Free Group Rings

In the case of free group rings the procedures of Section 2.1 can be replaced by procedures especially adapted to a special presentation of free groups leading to Procedure 10. For a free group $\mathcal{F}$ 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 the saturating set of the polynomial p is either of the form $\{\lambda\}$ (if p consists of one monomial only) or consists 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 [19, 14] for more details.

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 (compare [21, 19]). 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, where $|F|$ denotes the number of polynomials of F. In Procedure 11 (see page 12) $|p|$ denotes the number of monomials of a polynomial p.

prefix.groebner_basis_of_right_ideal_fg

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{F}]$.
Find: G, the reduced prefix Gröbner basis of $\mathsf{ideal}_r(F)$.

$H := \bigcup_{f \in F} \mathsf{prefix.saturate\_fg}(f)$; (*)
$G := \emptyset$;
while $H \neq \emptyset$ **do**
  $H := H \backslash \{can(p), acan(p)\}$;
  if $\mid can(p) \mid = 1$ **OR** $\mid acan(p) \mid = 1$
    then % the ideal is trivial
      $G := \{\lambda\}$;
      $H := \emptyset$;
    else $nf := \mathsf{normal.form}(can(p), \longrightarrow^{\mathrm{p}}_{G \cup H})$;
      if $\mid nf \mid = 1$
        then % the ideal is trivial
          $G := \{\lambda\}$;
          $H := \emptyset$;
        elseif $nf < can(p)$
          then $G := G \cup \mathsf{prefix.saturate\_fg}(nf)$;
          else $nf := \mathsf{normal.form}(acan(p), \longrightarrow^{\mathrm{p}}_{G \cup H})$;
            if $\mid nf \mid = 1$
              then % the ideal is trivial
                $G := \{\lambda\}$;
                $H := \emptyset$;
              elseif $nf < acan(p)$
                then $G := G \cup \mathsf{prefix.saturate\_fg}(nf)$;
                else $G := G \cup \{can(p), acan(p)\}$
              endif
        endif
  endif
  if $H = \emptyset$ **AND** a reduction occurred
    then $H := G$;
      $G := \emptyset$;
  endif
endwhile

Procedure 11: Prefix Gröbner basis computation for right ideals in free group rings

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 implemented 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

extend.left_fg

Given: A finite set $H \subseteq \mathbb{Q}[\mathcal{F}]$.
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 $G := G \cup$ prefix.saturate_fg(nf);
    endif
  endfor
endfor

Procedure 12: Preprocessing the set of polynomials

prefix.groebner_basis_of_two_sided_ideal_fg

Given: A finite set $F \subseteq \mathbb{Q}[\mathcal{F}]$.
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);
$G :=$ prefix.modified_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

Procedure 13: Prefix Gröbner basis computation for two-sided ideals in free group rings

saturated. We present an adaption of this idea for the case of free group rings in Procedure 12.

Procedure 13 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 (*). However, this procedure may not terminate.

2.5 The Special Case of Plain Group Rings

Plain groups are finite free products of free and finite groups. They can be presented by convergent string rewriting systems such that each generator has an inverse of length 1 and all rules are 2-monadic, i.e. for $\ell \longrightarrow r$ we have $|\ell| \leq 2$ and $|r| \leq 1$. Then the saturation Procedure 14 (see page 14) can be used to replace the

prefix.saturate_plain_group

Given: A polynomial $f \in \mathbb{Q}[\mathcal{P}]$,
R the set of rules of the presentation.
Find: S, a prefix saturating set for f.

$H :=$ prefix.saturate_fg(f);
$S := H$;
for all $h \in H$ **do**
  for all $(ab, c) \in R$ or $(ab, \lambda) \in R$ **do**
    if last$(HT(h)) = a$
      then $S := S \cup \{h * b\}$;
    endif
  endfor
endfor

Procedure 14: Prefix saturation in plain group rings

general one in the procedure prefix.groebner_basis_of_right_ideal_1 or in the procedure weakly.prefix.saturated_check used by the procedure prefix.groebner_basis_of_right_ideal_2 in Section 2.1.

Using this presentation of plain groups and the above saturation procedure, the procedures prefix.groebner_basis_of_right_ideal_1 and prefix.groebner_basis_of_right_ideal_2 terminate. Two sided-ideals can be computed the same way in combining the algorithms of Section 2.1 with the specialized saturation procedure. However, this procedure may not terminate.

2.6 The Special Case of Context-Free Group Rings

A finitely generated context-free group $\mathcal{G}$ is a group with a free normal subgroup of finite index. Hence the following presentations of context-free groups are used: Let the group $\mathcal{G}$ be given by X a finite set of generators for the free subgroup $\mathcal{F}$ and $\mathcal{E}$ the finite group such that $(\mathcal{E}\backslash\{\lambda\}) \cap X = \emptyset$ and $\mathcal{G}/\mathcal{F} \cong \mathcal{E}$. For all $e \in \mathcal{E}$ let $\phi_e : X \cup X^{-1} \longrightarrow \mathcal{F}$ be a function such that ϕ_λ is the inclusion and for all $x \in X \cup X^{-1}$, $\phi_e(x) = e^{-1} \circ_\mathcal{G} x \circ_\mathcal{G} e$. For all $e_1, e_2 \in \mathcal{E}$ let $z_{e_1,e_2} \in \mathcal{F}$ such that $z_{e_1,\lambda} \equiv z_{\lambda,e_1} \equiv \lambda$ and for all $e_1, e_2, e_3 \in \mathcal{E}$ with $e_1 \circ_\mathcal{E} e_2 =_\mathcal{E} e_3$, $e_1 \circ_\mathcal{G} e_2 \equiv e_3 z_{e_1,e_2}$. Let $\Sigma = (\mathcal{E}\backslash\{\lambda\}) \cup X \cup X^{-1}$ and let T contain the following rules:

$$
\begin{array}{lcll}
xx^{-1} & \longrightarrow & \lambda & \text{and} \\
x^{-1}x & \longrightarrow & \lambda & \text{for all } x \in X, \\
e_1e_2 & \longrightarrow & e_3 z_{e_1,e_2} & \text{for all } e_1, e_2 \in \mathcal{E}\backslash\{\lambda\}, e_3 \in \mathcal{E} \text{ such that } e_1 \circ_\mathcal{E} e_2 =_\mathcal{E} e_3, \\
xe & \longrightarrow & e\phi_e(x) & \text{and} \\
x^{-1}e & \longrightarrow & e\phi_e(x^{-1}) & \text{for all } e \in \mathcal{E}\backslash\{\lambda\}, x \in X.
\end{array}
$$

(Σ, T) then is convergent and is called a virtually free presentation of $\mathcal{G}$ (compare [5]). The elements of the group have representations of the form eu where $e \in \mathcal{E}$ and

prefix.saturate_context_free_group

Given: A polynomial $f \in \mathbb{Q}[\mathcal{C}]$,
Find: S, a prefix saturating set for f.

if $\mid f \mid = 1$
  then $S := \{\lambda\}$;
  else $ht := \mathsf{HT}(f)$;
    $acan(f) := f$;
    while $ht = \mathsf{HT}(acan(f))$ **and** $\mathsf{HT}(acan(f)) \notin \mathcal{E}$ **do**
      $can(f) := acan(f)$;
      $\sigma := \mathsf{last}(ht)^{-1}$;
      $ht := ht * \sigma$;
      $acan(f) := acan(f) * \sigma$;
    endwhile
    $S := \{can(f), acan(f)\}$;
endif

Procedure 15: Prefix saturation in context-free group rings

$u \in \mathcal{F}$. We can specify a total well-founded ordering on the group by combining a total well-founded ordering $\succeq_{\mathcal{E}}$ on $\mathcal{E}$ and a length-lexicographical ordering $\geq_{\text{lex}}$ on $\mathcal{F}$: Let $w_1, w_2 \in \mathcal{G}$ such that $w_i \equiv e_i u_i$ where $e_i \in \mathcal{E}$, $u_i \in \mathcal{F}$. Then we define $w_1 \succ w_2$ if and only if $|w_1| > |w_2|$ or ($|w_1| = |w_2|$ and $e_1 \succ_{\mathcal{E}} e_2$) or ($|w_1| = |w_2|$, $e_1 =_{\mathcal{E}} e_2$ and $u_1 >_{\text{lex}} u_2$). This ordering is compatible with right concatenation using elements in $\mathcal{F}$ in the following sense: Given $w_1, w_2 \in \mathcal{G}$ presented as described above, $w_1 \succ w_2$ implies $w_1 u \succ w_2 u$ for all $u \in \mathcal{F}$ in case $w_1 u, w_2 u \in \mathcal{G}$.

Using this presentation the saturation Procedure 15 can be used to replace the general one in the procedure prefix.groebner_basis_of_right_ideal_1 or in the procedure weakly.prefix.saturated_check used by the procedure prefix.groebner_basis_of_right_ideal_2 in Section 2.1. Additionally the procedures have to be extended by a preprocessing step on the input, namely each polynomial of the generating set has to be multiplied from the right with each character of the alphabet of the finite group presentation (see [19, 14].

Two sided-ideals can be computed the same way in combining the algorithms of Section 2.1 with the specialized saturation procedure. However, this procedure may not terminate.

2.7 Todd-Coxeter Simulation

One very popular procedure in combinatorial group theory is due to Todd and Coxeter and systematically enumerates all cosets of a finitely generated subgroup in a given finitely presented group [23]. Nielsen reduced sets allow the computation of Schreier coset representatives hence enabling syntactical solutions to the subgroup problem in finitely generated free groups [18]. In [20] Procedure 16 (see page 16) for simulating a combination of both procedures using prefix Gröbner bases is developed. It makes use

extended_todd_coxeter_simulation

Given: $F_R = \{r - 1 \mid r \in R\}$, a set of binomials representing the relators.
$F_U = \{u - 1 \mid u \in U\}$, a set of binomials representing the subgroup generators.

Find: If $F_R = \emptyset$ a Nielsen reduced generating set for the subgroup is generated.
Else the cosets of the subgroup in the group are enumerated and on
termination a multiplication table is represented by the prefix Gröbner basis.

$N := \emptyset$;
if $F_R = \emptyset$
  then $G :=$ prefix.groebner_basis_of_right_ideal_fg(F_U);
  else $N := \{\lambda\}$;
    $B := \{a \mid a \in \Sigma \cup \Sigma^{-1}\}$;
    $G :=$ prefix.groebner_basis_of_right_ideal_fg$(F_R \cup F_U)$;
    while $B \neq \emptyset$ **do**
      $\tau := \min_<(B)$;
      $B := B \backslash \{\tau\}$;
      if τ is not prefix reducible by G
        then $N := N \cup \{\tau\}$;
          $B := B \cup \{\tau a \mid a \in (\Sigma \cup \Sigma^{-1}) \backslash \{(\ell(\tau))^{-1}\}\}$;
          $H := \{\tau * (r - 1) \mid r - 1 \in F_R\}$;
          $G :=$ prefix.groebner_basis_of_right_ideal_fg$(G \cup H)$;
          $S := \{w \in N \mid w$ is prefix reducible by $G\}$;
          $N := N \backslash S$;
      endif
    endwhile
endif

Procedure 16: Extended Todd-Coxeter simulation

of the fact that group presentations as well as subgroup generators can be represented by binomial polynomial sets in the free group ring over the generators of the group. If the set of group relators is empty, the procedure always terminates and produces a Nielsen reduced set for the subgroup represented by G. If the set of relators is not empty, the procedure on termination provides the coset representatives of the subgroup in N and the non-trivial part of the multiplication table with group generators in G. Hence in this case it terminates if the submonoid has finite index.

3 MRC V 1.0

This section describes MRC V 1.0, the **M**onoid **R**ing **C**ompletion program. First, we give an overview of the procedures implemented in Section 3.1. Then we will present the commandline syntax and the file formats in Section 3.2. Finally, we will three small examples of the use and the output of MRC V 1.0 in Section 3.3.

3.1 Provided procedures

3.1.1 Prefix Gröbner Bases of Right Ideals

MRC V 1.0 provides the following procedures for the computation of prefix Gröbner bases of right ideals:

GB	prefix.groebner_basis_of_right_ideal_1 as mentioned in Section 2.1
GBIR	prefix.groebner_basis_of_right_ideal_1 as mentioned in Section 2.1 followed by interreduction of the resulting Gröbner Base
IRGB	prefix.groebner_basis_of_right_ideal_2 as mentioned in Section 2.1
ENUM	prefix.groebner_basis_of_right_ideal_enumerate as mentioned in Section 2.2
FM	prefix.groebner_basis_of_right_ideal_fm as mentioned in Section 2.3
FG	prefix.groebner_basis_of_right_ideal_fg as described in Section 2.4
PG	prefix.groebner_basis_of_right_ideal_pg as described in Section 2.5
CFG	prefix.groebner_basis_of_right_ideal_cfg as described in Section 2.6

The input required for these procedures consists of 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 interreduced string rewriting system. The specialized algorithms for free monoids and free, plain and context-free groups require the special presentations as described in the respective sections. In general, a convergent string rewriting system 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 V 1.0. The result produced by MRC V 1.0 is a set of polynomials representing the respective prefix Gröbner basis.

3.1.2 Prefix Gröbner Bases of Two-Sided Ideals

MRC V 1.0 provides the following procedures for the computation of prefix Gröbner bases of two-sided ideals:

ZI	prefix.groebner_basis_of_two_sided_ideal as described in Section 2.1
ZIFM	prefix.groebner_basis_of_two_sided_ideal_fm as mentioned in Section 2.3
ZIFG	prefix.groebner_basis_of_two_sided_ideal_fg as described in Section 2.4
ZIPG	prefix.groebner_basis_of_two_sided_ideal_pg as mentioned in Section 2.5
ZICFG	prefix.groebner_basis_of_two_sided_ideal_cfg as described in Section 2.6

The procedures for computing prefix Gröbner bases of two-sided ideals, too, take a

monoid presentation and a set of polynomials as input. The specialized procedures for free monoids and free, plain and context-free groups require the special presentations as described in the respective sections. 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.

3.1.3 Other Methods

In addition MRC V 1.0 provides the following procedures:

NF	normal_form computation of normal forms for each polynomial of a set of polynomials with respect to a given set of generators of an ideal
MEMBER	a semi-decision procedure to decide whether a given set of polynomials lies in an ideal, the ideal is enumerated and after each step it is checked whether the polynomials can be reduced to zero by the part of the Gröbner basis so far enumerated or not
IR	prefix_interreduce to prefix interreduce a set of polynomials as described in Section 2.1
TC	extended_todd_coxeter_simulation as described in Section 2.7

The normal form procedure expects as input a monoid presentation, a set of polynomials generating an ideal (generators), and a set of polynomials of which the normal form shall be computed. The output consists of the polynomials and their respective normal form with respect to the generating set.

The member procedure, too, expects as input a monoid presentation, a set of polynomials generating an ideal (generators), and a set of polynomials of which the normal-form shall be computed. The output consists of the part of the Gröbner basis so far enumerated and those polynomials which can be reduced to zero by this part. If the enumeration process terminates then the resulting Gröbner basis is printed and the polynomials not lying in the Gröbner basis are identified as such.

The interreduction procedure expects as input a monoid presentation and a set of polynomials. The output is the interreduced set of polynomials. For polynomials from the free monoid ring as presented in Section 2.3 this is equivalent to computing the reduced prefix Gröbner basis.

The Todd-Coxeter simulation procedure expects as input a free group presentation, a set of polynomials generating a group, and a set of polynomials generating a subgroup of the group. The output consists either of the Nielsen reduced set or of the coset representatives and a Gröbner basis representing the non-trivial part of the multiplication table.

3.1.4 Other information provided

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

3.2 Unix Commandline Syntax and File Formats

The call to MRC V 1.0 has one of the following forms:

```
mrc            NF        monoid-presentation set-of-generators set-of-polynomials
mrc            MEMBER    monoid-presentation set-of-generators set-of-polynomials
mrc            IR        monoid-presentation set-of-generators

mrc            GB        monoid-presentation set-of-generators
mrc            GBIR      monoid-presentation set-of-generators
mrc            IRGB      monoid-presentation set-of-generators
mrc            ENUM      monoid-presentation set-of-generators

mrc            FM        fm-presentation set-of-generators
mrc            FG        fg-presentation set-of-generators
mrc            PG        pg-presentation set-of-generators
mrc            CFG       cfg-presentation set-of-generators

mrc  [-batch]  ZI        monoid-presentation set-of-generators
mrc  [-batch]  ZIFM      fm-presentation set-of-generators
mrc  [-batch]  ZIFG      fg-presentation set-of-generators
mrc  [-batch]  ZIPG      pg-presentation set-of-generators
mrc  [-batch]  ZICFG     cfg-presentation set-of-generators

mrc            TC        fg-presentation set-of-relators-group set-of-relators-subgroup
```

The program name is followed by the name of the method, the name of the file containing the monoid presentation, and one or more names of files containing a set of polynomials each. Optionally the flag "-batch" can be provided. The output produced depends on the parameters chosen, in each case the input and output values of the methods described in the previous section are written to the terminal.

The file format of the respective monoid presentation is given by the following grammar:

```
CHARACTER  =  "(" letter " " weight ")"
ALPHABET   =  {CHARACTER}*

ORDERING   =  "length-lexicographic"

TERM       =  {letter}* | "$\lambda$"
RULE       =  "(" TERM " " TERM ")"
RULESET    =  {RULE}*
```

MONOID	=	ALPHABET ";" ORDERING ";" RULESET ";"
FM	=	ALPHABET ";" ORDERING ";"
FG	=	ALPHABET ";" ALPHABET ";" ORDERING ";"
PG	=	ALPHABET ";" ALPHABET ";" ORDERING ";" RULESET ";"
CFG	=	ALPHABET ";" ALPHABET ";" ALPHABET ";" ORDERING ";" RULESET ";"

where "letter" is an ASCII-value and "weight" is a positive integer value. The weight component is used for realizing the ordering on the monoid. MONOID, FM, FG, PG, CFG are the file formats for the respective presentations.

The ruleset for the free monoid (FM) is empty. The second alphabet of the free and plain group (FG, PG) gives the inverse characters, that is, it consists of the same characters but ordered such that at each position stands the inverse character to the one at the same position of the first alphabet. The set of rules of the free group is empty. The rules are generated from the two alphabets. The first alphabet of the context-free group (CFG) denotes the finite group part, the second and the third alphabet the free group part.

No syntactical or semantical check is provided. It is supposed that the presentations are conform to the requirements for the respective monoids and groups.

For example the free group with one generator can be presented as a monoid as follows where a is the generator, A its formal inverse and `$\lambda$` decodes the trivial element of $\mathcal{G}$:

```
( A 4 ) ( a 3 );
length-lexicographic ;
( aA  $\lambda$ )
( Aa  $\lambda$ );
```

The same free group will be presented as a free group as follows.

```
( A 4 ) ( a 3 );
( a 3 ) ( A 4 );
length-lexicographic ;
```

The file format of a set of polynomials is given by the following grammar:

DIGIT	=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
INTEGER	=	{DIGIT}+
COEFF	=	["-"]INTEGER "/" INTEGER
TERM	=	{letter}* | "`$\lambda$`"
MONOM	=	COEFF "*" TERM
POLYNOM	=	"(" MONOM {"+" MONOM}* ")"
POLYSET	=	{POLYNOM}* ";"

For example the set of polynomials $aaaaaa - 1$ and $aaa - 1$ is presented as follows:

```
( 1 / 1 * aaaaaa + -1 / 1 * $\lambda$ )
( 1 / 1 * aaa + -1 / 1 * $\lambda$ ) ;
```

Set-of-generators, set-of-relators-group and set-of-relators-subgroup are sets of polynomials where set-of-generators means the set of polynomials generating an ideal.

In order to get an input which will be accepted by MRC V 1.0 there exists a program named "filter" which transforms a file following one of the above grammars into an input file for MRC V 1.0. The call to this program is: "mrc-filter <input >output".

3.3 Examples

3.3.1 Ideal Membership Problem

Let the plain group $\mathcal{P}$ be presented by $(a, B, b, C, c, d; aa = Bb = bB = Cc = cC = dd = 1, ab = c, ac = b, Ba = C, Bc = d, bC = a, bd = c, Ca = B, Cb = d, cB = a, cd = b, dB = C, dC = B)$. Now we want to decide whether the polynomials $a+b+c$, $ad + a + 1$ and $bc + c^2 + b$ lie in the ideal generated by the polynomial $a + b + c$.

```
You have chosen the following parameters.

Monoid:
Alphabet:
(a 6) (B 5) (b 4) (C 3) (c 2) (d 1) ;
Ordering:
length-lexicographic ;
Rules:
(aa $\lambda$)
(ab c)
(ac b)
(Ba C)
(Bb $\lambda$)
(Bc d)
(bB $\lambda$)
(bC a)
(bd c)
(Ca B)
(Cb d)
(Cc $\lambda$)
(cB a)
(cC $\lambda$)
(cd b)
(dB C)
(dC B)
(dd $\lambda$)

Generating set of polynomials:
(1/1 * a + 1/1 * b + 1/1 * c)

Set of polynomials to test:
(1/1 * ad + 1/1 * a + 1/1 * $\lambda$)
(1/1 * a + 1/1 * b + 1/1 * c)
(1/1 * bc + 1/1 * cc + 1/1 * b)
```

```
Method: (MEMBER) test ideal membership while enumerating a prefix Groebner
basis of a right ideal in a monoid ring

Number of polynomials still to saturate: 1
So far enumerated polynomials
(1/1 * a + 1/1 * b + 1/1 * c)
(1/1 * ba + 1/1 * ca + 1/1 * $\lambda$)
(1/1 * bb + 1/1 * cb + 1/1 * c)
(1/1 * bc + 1/1 * cc + 1/1 * b)

1/1 * a + 1/1 * b + 1/1 * c is member of the right ideal
1/1 * bc + 1/1 * cc + 1/1 * b is member of the right ideal

Number of polynomials still to saturate: 1
So far enumerated polynomials
(1/1 * a + 1/1 * b + 1/1 * c)
(1/1 * ba + 1/1 * ca + 1/1 * $\lambda$)
(1/1 * bb + 1/1 * cb + 1/1 * c)
(1/1 * bc + 1/1 * cc + 1/1 * b)


Number of polynomials still to saturate: 1
So far enumerated polynomials
(1/1 * a + 1/1 * b + 1/1 * c)
(1/1 * ba + 1/1 * ca + 1/1 * $\lambda$)
(1/1 * bb + 1/1 * cb + 1/1 * c)
(1/1 * bc + 1/1 * cc + 1/1 * b)


saturator deleted!
Number of polynomials still to saturate: 0
So far enumerated polynomials
(1/1 * a + 1/1 * b + 1/1 * c)
(1/1 * ba + 1/1 * ca + 1/1 * $\lambda$)
(1/1 * bb + 1/1 * cb + 1/1 * c)
(1/1 * bc + 1/1 * cc + 1/1 * b)


1/1 * ad + 1/1 * a + 1/1 * $\lambda$ is not member of the right ideal

Memory Usage: 2441216
```

The result is that $a+b+c$ and $bc+c^2+b$ obviously lie in the ideal, whereas $ad+a+1$ does not.

3.3.2 Todd Coxeter 1

There is a family of presentations of trivial groups due to B. H. Neumann, who suggested that the members of the family provide a challenge for coset enumeration programs (see [22]). While Sims used the completion procedure for string rewriting systems due to Knuth and Bendix to study the first member of the family, we show that this can also be done using the algorithm presented in Section 2.7.

The first presentation of the trivial group is $RsrSS = StsTT = TrtRR = 1$, where r, s, t are the generators and R, S, T denote the respective inverses. The generating set of the subgroup is empty. MRC V 1.0 produces the following output.

```
Group:
Alphabet:
(r 1) (R 2) (s 3) (S 4) (t 5) (T 6) ;
(R 2) (r 1) (S 4) (s 3) (T 6) (t 5) ;
Ordering:
length-lexicographic ;
Ruleset:
(rR $\lambda$)
(Rr $\lambda$)
(sS $\lambda$)
(Ss $\lambda$)
(tT $\lambda$)
(Tt $\lambda$)

Generating set of polynomials (Relators):
(1/1 * RsrSS + -1/1 * $\lambda$)
(1/1 * StsTT + -1/1 * $\lambda$)
(1/1 * TrtRR + -1/1 * $\lambda$)
Generating set of polynomials (Subgroup):
Method: (TC) Todd-Coxeter enumeration of cosets

Coset representatives:
{ $\lambda$ }
Interreduced prefix Groebner basis:
(1/1 * r + -1/1 * $\lambda$)
(1/1 * R + -1/1 * $\lambda$)
(1/1 * s + -1/1 * $\lambda$)
(1/1 * S + -1/1 * $\lambda$)
(1/1 * t + -1/1 * $\lambda$)
(1/1 * T + -1/1 * $\lambda$)

Memory Usage: 3883008
```

MRC V 1.0 needed 816 seconds and 4 MB memory to compute this result on a Sparc Ultra 177 running Solaris 2.5.

The second member of the family is far more difficult to compute using this method. It was not possible within using the resources (time and memory) available. Using a different strategy improves the run-time needed but still the memory requirements are too high ($>$ 200 MB memory).

3.3.3 Todd Coxeter 2

The second example for the computation of cosets using the procedure of Section 2.7 is the Dyck group D(2,3,2) presented by $\Sigma = \{a, b\}$ and $R = \{aaa, bbb, abab\}$ and $\mathcal{U}$ the subgroup of $\mathcal{G}$ generated by $\{a\}$.

```
Group:
Alphabet:
(A 4) (a 3) (B 2) (b 1) ;
(a 3) (A 4) (b 1) (B 2) ;
Ordering:
length-lexicographic ;
Ruleset:
(Aa $\lambda$)
(aA $\lambda$)
(Bb $\lambda$)
(bB $\lambda$)

Generating set of polynomials (Relators):
(1/1 * aaa + -1/1 * $\lambda$)
(1/1 * abab + -1/1 * $\lambda$)
(1/1 * bbb + -1/1 * $\lambda$)
Generating set of polynomials (Subgroup):
(1/1 * a + -1/1 * $\lambda$)
Method: (TC) Todd-Coxeter enumeration of cosets

Coset representatives:
{ $\lambda$, b, B, bA }
Interreduced prefix Groebner basis:
(1/1 * A + -1/1 * $\lambda$)
(1/1 * a + -1/1 * $\lambda$)
(1/1 * BA + -1/1 * b)
(1/1 * Ba + -1/1 * bA)
(1/1 * BB + -1/1 * b)
(1/1 * bAA + -1/1 * B)
(1/1 * bAB + -1/1 * bA)
(1/1 * bAb + -1/1 * bA)
(1/1 * ba + -1/1 * B)
(1/1 * bb + -1/1 * B)


Memory Usage: 2424832
```

MRC V 1.0 gives the coset representatives $\{\lambda, b, B, bA\}$ and the non-trivial part of the multiplication table $\{A \to \lambda, a \to \lambda, BA \to b, Ba \to bA, BB \to b, bAA \to B, bAB \to bA, bAb \to bA, ba \to B, bb \to B\}$ using 2.5 MB memory and less than 1 second cpu time on a Sparc Ultra 177 running Solaris 2.5.

4 Implementation

This section explains the rationales behind the implementation. In Section 4.1 goals and the general structure of the system are described. The differences to commutative polynomial rings described in Section 4.2 motivate the necessity of data structures adjusted to prefix reduction which are explained in Section 4.3.

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 chosen 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 V 1.0 is implemented in C++ using the LEDA-Library on SUN SparcWorkstations under Solaris 2.5.

Based on the procedures in [14] 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 led to an architecture whose general structure is depicted in Figure 1 (see page 26). Only the module names and the data sections are shown, the methods have been omitted.

This diagram shows that based on the mathematical foundation of the theory, classes were built to represent the mathematical entities COEFFicient, TERM, and POLYNOMial where the coefficients and terms are joined to form polynomials. The terms are formed with respect to the MONOID. The other concepts depicted in the top level class diagram arise from the algorithms and represent sets of other objects or procedures which can be separated out. There exist additional diagrams not reproduced here for the concepts 'monoid', 'polyset' and 'ideal'.

The abstract procedures of [14] 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.

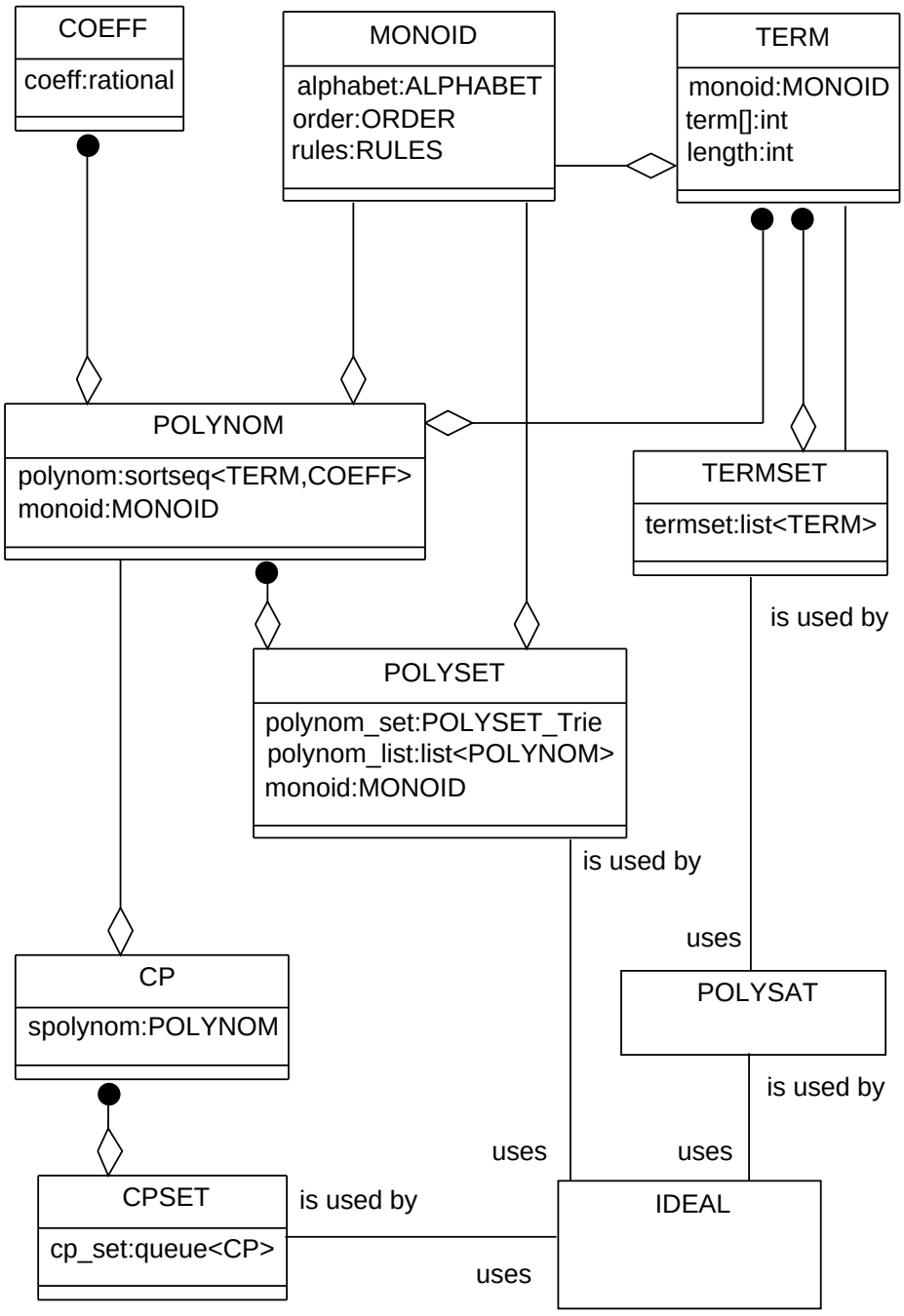

Figure 1: System structure

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.

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 implementation of 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 and can be seen as sequences of characters. In order to reduce one term with another term 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 terms 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 are 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 in [9], 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 becomes 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.

An example is given in Figure 2 where the following set of polynomials S is stored: $\Sigma = \{a, b, c\}, a > b > c, T = \{bc \to \lambda\}$, $S = \{\underline{a} - c, \underline{aa} - bb, \underline{ac} - ba, \underline{b} - c, \underline{cb} - b, \underline{cc} - a\}$.

4.3.2 Ternary Search Tree

Ternary search trees as implemented in MRC V 1.0 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.

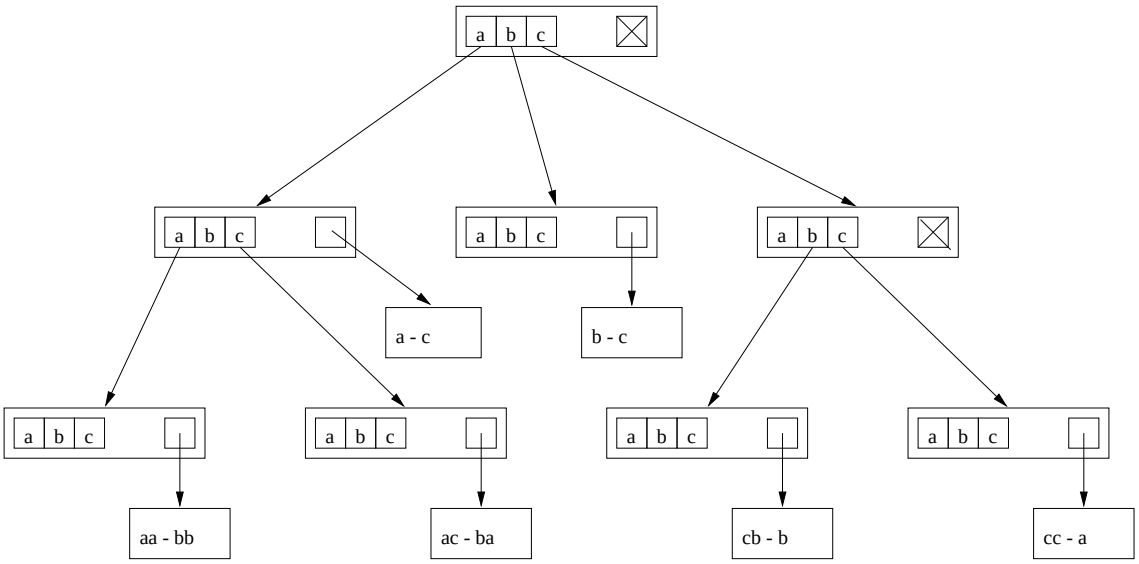

Figure 2: Example of a trie

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:

- 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 of the term becomes 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 of the term becomes 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.

Deletion and lookup of keys follow the same scheme.

An example is given in Figure 3 where the following set of polynomials S is stored (the same as in Section 4.3.1): $\Sigma = \{a, b, c\}, a > b > c, T = \{bc \rightarrow \lambda\}$, $S = \{\underline{a} - c, \underline{aa} - bb, \underline{ac} - ba, \underline{b} - c, \underline{cb} - b, \underline{cc} - a\}$. Note, that the actual structure of the ternary search tree depends on the order in which its elements were inserted.

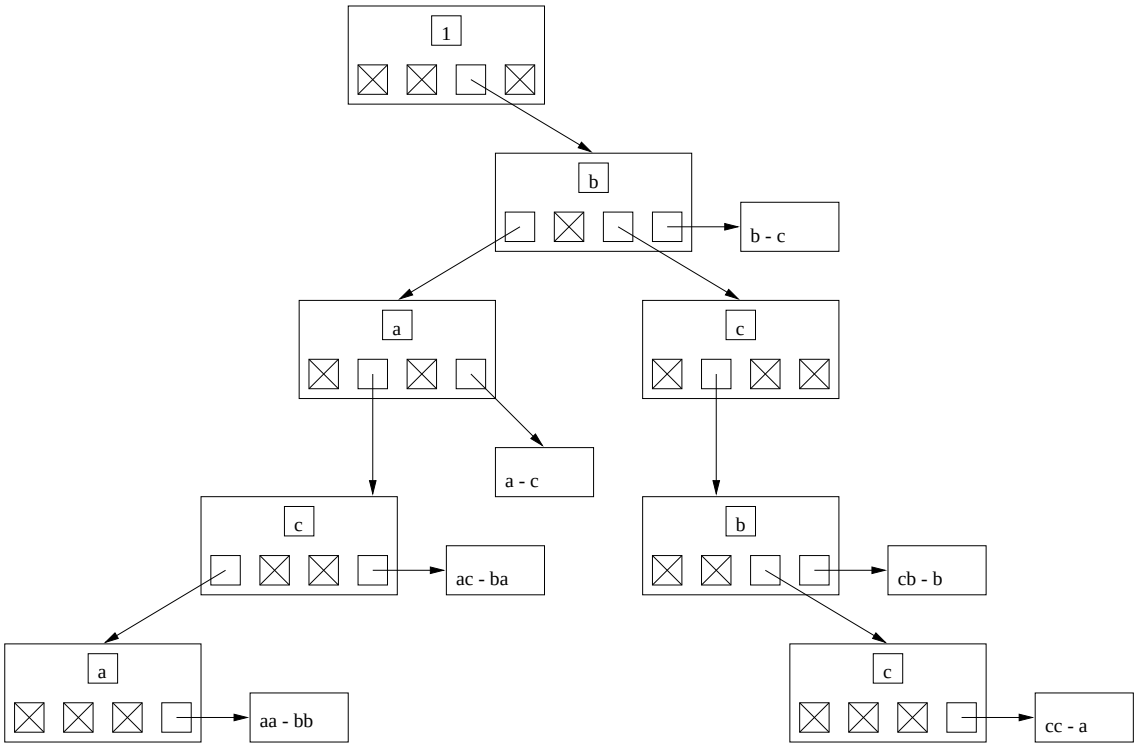

Figure 3: Example of a ternary search tree

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. Both data structures need the same number of nodes storing the same set of keys. Another disadvantage of tries is that changing the alphabet requires a total reorganization of the trie structure.

Practical tests showed the following phenomenon. There is an example where no difference of the runtime was observed for the computation of the Gröbner basis, but 16 % less space was used using the ternary search tree compared to using 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.

Enumerating all elements of the trie will in general be slower than enumerating all elements of the ternary search tree (if the same keys are stored). This is due to the fact that in general less pointers have to be considered for the ternary search tree compared with the trie.

5 Enhancements

Having implemented procedures for treating right ideals of monoid rings using the concept of prefix reduction several enhancements are possible now. On the one hand monoid rings over reduction rings especially the integers can be implemented (see [13]). On the other hand, procedures for other specialized reduction concepts like commutative reduction (for commutative monoids) and quasi-commutative reduction (for polycyclic groups) can be added to the system.

Further there is a potential for making enhancements concerning the time and space consumption of the existing procedures. Therefore statistical information gained from comparison runs is evaluated. The ternary search tree is one such enhancement already realized. Other possibilities are the use of 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 Kaiserslautern 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 and saturation). 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 and W. Lassner. An extension of Buchberger's algorithm and calculations in enveloping fields of Lie algebras. *Journal of Symbolic Computation*, 6:361–370, 1988.

[3] J. Bently and 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] R. Cremanns and F. Otto. Constructing canonical presentations for subgroups of context-free groups in polynomial time. In J. von zur Gathen and M. Giesbrecht, editors, *Proc. ISSAC'94*, pages 147–153. ACM, 1994.

[6] A. Kandri-Rody and V. Weispfenning. Non-commutative Gröbner bases in algebras of solvable type. *Journal of Symbolic Computation*, 9:1–26, 1990.

[7] B. J. Keller. Alternatives in implementing noncommutative Gröbner basis systems. In *Proceedings of the Workshop on Symbolic Rewriting Techniques, Monte Verita, 1995*, pages 127–180. Birkhäuser, 1998.

[8] W. Lassner. Symbol representations of noncommutative algebras. In *EUROCAL'85*, LNCS 204, pages 99–115. Springer, 1985.

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

[10] K. Madlener and B. Reinert. Computing Gröbner bases in monoid and group rings. In M. Bronstein, editor, *Proc. ISSAC'93*, pages 254–263. ACM, 1993.

[11] K. Madlener and B. Reinert. A generalization of Gröbner bases algorithms to nilpotent group rings. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):103–123, 1997.

[12] K. Madlener and B. Reinert. A generalization of Gröbner basis algorithms to polycyclic group rings. *Journal of Symbolic Computation*, 25(1):23–45, 1998.

[13] K. Madlener and B. Reinert. Gröbner bases in non-commutative reduction rings. In B. Buchberger and F. Winkler, editors, *Gröbner Bases and Applications (Proc. of the Conference 33 Years of Gröbner Bases)*, volume 251 of *London Mathematical Society Lecture Notes Series*, pages 408–420. Cambridge University Press, 1998.

[14] K. Madlener and B. Reinert. String rewriting and Gröbner bases – a general approach to monoid and group rings. In M. Bronstein, J. Grabmeier, and V. Weispfenning, editors, *Proceedings of the Workshop on Symbolic Rewriting Techniques, Monte Verita, 1995*, volume 15 of *Progress in Computer Science and Applied Logic*, pages 127–180. Birkhäuser, 1998.

[15] K. Madlener and B. Reinert. Relating rewriting techniques on monoids and rings: Congruences on monoids and ideals in monoid rings. *Theoretical Computer Science*, to appear.

[16] F. Mora. Gröbner bases for non-commutative polynomial rings. In *Proc. AAECC-3*, LNCS 229, pages 353–362. Springer, 1985.

[17] T. Mora. An introduction to commutative and non-commutative Gröbner bases. *Theoretical Computer Science*, 134:131–173, 1994.

[18] J. Nielsen. Om Regning med ikke kommutative Faktoren og dens Anvendelse i Gruppeteorien. *Mat. Tidsskr. B.*, pages 77–94, 1921.

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

[20] B. Reinert, T. Mora, and K. Madlener. A note on Nielsen reduction and coset enumeration. In *Proc. ISSAC'98*, 1998.

[21] A. Rosenmann. An algorithm for constructing Gröbner and free Schreier bases in free group algebras. *Journal of Symbolic Computation*, 16:523–549, 1993.

[22] C. Sims. The Knuth-Bendix procedure for strings as a substitute for coset enumertation. *Journal of Symbolic Computation*, 12:439–442, 1991.

[23] J. Todd and H. Coxeter. A practical method for enumerating cosets of a finite abstract group. In *Proc. Edinburgh Math. Soc.*, volume 5, pages 26–34, 1936.

A Proofs

The proofs are based on the results of Section 4.4 in [19]. They use the equivalence of prefix Gröbner bases and prefix standard bases.

Definition 8 *Let F be a set of polynomials and p a non-zero polynomial in $\mathbb{K}[\mathcal{M}]$. A representation*

$$p = \sum_{i=1}^{n} \alpha_i \cdot f_i * w_i, \text{ with } \alpha_i \in \mathbb{K}^*, f_i \in F, w_i \in \mathcal{M}$$

is called a **prefix standard representation** *with respect to the set of polynomials F, in case for all $1 \leq i \leq n$ we have $\mathsf{HT}(p) \succeq \mathsf{HT}(f_i)w_i$. A set $F \subseteq \mathbb{K}[\mathcal{M}]$ is called a* **prefix standard basis** *if every non-zero polynomial $g \in \mathsf{ideal}_r(F)$ has a prefix standard representation with respect to F.*

Note, that $\mathsf{HT}(p) \equiv \mathsf{HT}(f_j)w_j$ occurs for at least one polynomial in this representation.

Theorem 9 *For a set $F \subseteq \mathbb{K}[\mathcal{M}]$, the following statements are equivalent:*

1. *F is a prefix Gröbner basis.*
2. *For all polynomials $g \in \mathsf{ideal}_r(F)$ we have $g \xrightarrow{*}{}^{\mathrm{P}}_{F} 0$.*
3. *F is a prefix standard basis.*

Theorem 10 *For a set F of polynomials in $\mathbb{K}[\mathcal{M}]$, equivalent are:*

1. *Every polynomials $g \in \mathsf{ideal}_r(F)$ has a prefix standard representation.*
2. *(a) For all polynomials $f \in F$ and all elements $w \in \mathcal{M}$, the polynomial $f * w$ has a prefix standard representation.*
   (b) For all polynomials $f_k, f_l \in F$ the non-trivial prefix s-polynomials have prefix standard representations.

Remark 11 *For a set of polynomials G in $\mathbb{K}[\mathcal{M}]$, if $G' = \mathsf{prefix.interreduce}(G)$, then*

1. *The procedure* prefix.interreduce *always terminates.*
2. $\mathsf{ideal}_r(G) = \mathsf{ideal}_r(G')$.
3. *If a polynomial g has a prefix standard representation with respect to G it also has one with respect to G'.*

Remark 12 *For $G \subseteq \mathbb{K}[\mathcal{M}]$,* $\mathsf{weakly.prefix.saturated\_check}(G) \subseteq \mathsf{ideal}_r(G)$.

These results can be combined to prove Theorem 4 from Section 2:

Theorem 4: *Let H be the set generated by procedure* weakly.prefix.saturated_check. *Then $H = \emptyset$ if and only if G is weakly prefix saturated.*

Proof: If $H = \emptyset$ then we can use Theorem 10 to show that each $g \in G$ has a prefix standard representation and thus G is in fact a prefix Gröbner basis by Theorem 9 and hence weakly prefix saturated. As G is a reduced set there are no prefix s-polynomials. It remains to show that for each $g \in G$ and $w \in \mathcal{M}$ the multiple $g * w$ has a prefix standard representation. Notice that the test set S for G contains a polynomial h such that $g * w \longrightarrow^{\mathrm{p}}_{h} 0$, i.e. $\mathsf{HT}(g * w) = \mathsf{HT}(h * u) \equiv \mathsf{HT}(h)u$ for some $u \in \mathcal{M}$. Moreover, as $H = \emptyset$ the polynomial h must have a prefix standard representation. But then so must $h * u$, namely the one of h multiplied by u.

On the other hand, let G be weakly prefix saturated. Since G is also reduced every normal form with respect to G is unique (no s-polynomials!). Hence every polynomial in the test set must reduce to zero and H must be empty. q.e.d.

Let $G_i =$ prefix.interreduce$(G_{i-1} \cup H_{i-1})$ and $H_i =$ weakly.prefix.saturated_check(G_i) be the respective sets generated by procedure prefix.groebner_basis_of_right_ideal_2 in the i-th iteration step. Then the following Lemmata hold (compare [19] Lemmata 4.4.45 and 4.4.46):

Lemma 13 *If $f \in G_k$ for some $k \in \mathbb{N}$, then there exists a polynomial $g \in G$ such that $\mathsf{HT}(g)$ is a prefix of $\mathsf{HT}(f)$.*

Proof: In case our procedure terminates or $f \in G$ we are done at once. Hence, let us assume there exists a polynomial f such that $f \in G_k$ for some $k \in \mathbb{N}$ but no $g \in G$ exists such that $\mathsf{HT}(g)$ is a prefix of $\mathsf{HT}(f)$. Further let f be a counter-example with minimal head term. As $f \notin G$ there exists an index $j > k$ such that $f \in G_{j-1}$ but $f \notin G_j$.

f can only be eliminated during the step $G_j =$ prefix.interreduce$(G_{j-1} \cup H_{j-1})$. That implies the existence of a polynomial $q \in G_j$ such that $\mathsf{HT}(f) \equiv \mathsf{HT}(q)w$ for some $w \in \mathcal{M}$. $w \neq \lambda$ would imply the existence of a smaller counter-example, as then $\mathsf{HT}(q) < \mathsf{HT}(f)$ and each prefix of $\mathsf{HT}(q)$ is also a prefix of $\mathsf{HT}(f)$. That is why $\mathsf{HT}(f) \equiv \mathsf{HT}(q)$ must hold. Without loss of generality q is the only polynomial with $\mathsf{HT}(f) \equiv \mathsf{HT}(q)$ which lies in the interreduced set and $q =$ normal.form$(f, G_j \backslash \{q\})$. All other polynomials must have head terms different from $\mathsf{HT}(q)$ as they would otherwise be reducible.

As prefix reduction is Noetherian we can conclude that there must exist some $n \in \mathbb{N}$ such that $f \overset{*}{\longrightarrow}^{\mathrm{p}} q_1 \overset{*}{\longrightarrow}^{\mathrm{p}} q_2 \ldots \overset{*}{\longrightarrow}^{\mathrm{p}} q_n$, $q_i \in G_{j-1+i}$, $\mathsf{HT}(f) \equiv \mathsf{HT}(q_i)$ for $1 \leq i \leq n$ and q_n is irreducible in all following steps. Thus, $q_n \in G$, $\mathsf{HT}(q_n) \equiv \mathsf{HT}(f)$ is a contradiction to our assumption. Notice that $\mathsf{HT}(f) > \mathsf{HT}(q_n)$ would give rise to a smaller counter-example and hence is not possible. q.e.d.

Corollary 14 *If $f \in G_i$, $f \notin G_{i+1}$, then there is no $g \in G_k$, $k > i+1$ such that $\mathsf{HT}(f)$ is a prefix of $\mathsf{HT}(g)$.*

Lemma 15 *Let G be the set generated by procedure* prefix.groebner_basis_of_right_ideal_2. *Then if $f \in H_k \cup G_k$ for some $k \in \mathbb{N}$, f has a prefix standard representation.*

Proof: This follows from the fact that every $f \in H_k \cup G_k$ has a prefix standard representation with respect to G_{k+1} and Lemma 13. q.e.d.

We can now proof the Theorems 5 and 6 from Section 2:

Theorem 5 *Let G be the set generated by procedure* prefix.groebner_basis_of_right_ideal_2 *on a finite input $F \subseteq \mathbb{K}[\mathcal{M}]$. Then G is a reduced prefix Gröbner basis.*

Proof: In case procedure prefix.groebner_basis_of_right_ideal_2 terminates we have $G = G_k$ for some $k \in \mathbb{N}$. Otherwise, we have $G = \bigcup_{i \geq 0} \bigcap_{j \geq i} G_j$.
By construction no prefix s-polynomials exist for the polynomials in G. Hence, in order to show that G is a Gröbner basis, by Theorem 10 it remains to show that $g \in G$ implies that for all $w \in \mathcal{M}$ the multiple $g * w$ has a prefix standard representation with respect to G. This follows immediately as in the proof of Theorem 4. q.e.d.

Let $G_i =$ prefix.interreduce$(G_{i-1} \cup H_{i-1})$ and $H_i =$ weakly.prefix.saturated_check(G_i).

Theorem 6 *Let $F \subseteq \mathbb{K}[\mathcal{M}]$ be a finite set of polynomials. In case* ideal$_r(F)$ *has a finite reduced prefix Gröbner basis, the procedure* prefix.groebner_basis_of_right_ideal_2 *terminates.*

Proof: Because of Corollary 14 no cycles occur during the computation of G. Further the number of polynomials which have a prefix standard representation with respect to G_{i+1} is greater than with respect to G_i for all i. Since reduced prefix Gröbner bases are unique up to multiplication with coefficients the existence of a finite one implies that all are finite. As this applies to the one computed by procedure prefix.groebner_basis_of_right_ideal_2 the computation then must terminate. q.e.d.

B Classes

In this section the classes which form MRC V 1.0 are listed. The class name is given in uppercase except for template classes where only the first character is uppercase and the rest lowercase. The data are given in `typewriter` style, the methods in *typewriter slanted*. This section together with the class diagrams 'mrc', 'monoid', 'polyset', and 'ideal' (which are not reproduced here) is meant as additional documentation for the source code and is incomprehensible otherwise.

First of all the prefix tree classes are described. The other classes are grouped in the same way as the source code.

B.1 Prefix trees

Prefix trees and their implementation as tries or ternary search trees were already described on an abstract level in Section 4.3. In these sections implementational details and the required methods of classes used for the instantiation of these template classes are described. Prefix trees will be abbreviated as PTREE. This name can be substituted by either TRIE or TTREE as described below.

B.1.1 Trie

A node of the trie consists of four elements:

- The `width` (n) of the TRIE. It contains the number of indices (internal representation of a CHARACTER) used.
- An array of pointers to the successor `elements`. The length of this list is `width`.
- The `content` (C). This is a pointer to a structure which contains the elements belonging to a key.
- The `number` of contents (number_of_contents) stored in the subtree rooted at this node.

In Figure 4 the general structure of a trie is depicted. An example (with n and number_of_contents omitted) is depicted in Figure 2 (see Section 4.3.1).

B.1.2 Ternary Search Tree

A node of the ternary search tree consists of six elements:

- The `key`. This is the letter stored in this node.
- A pointer to the `left`.

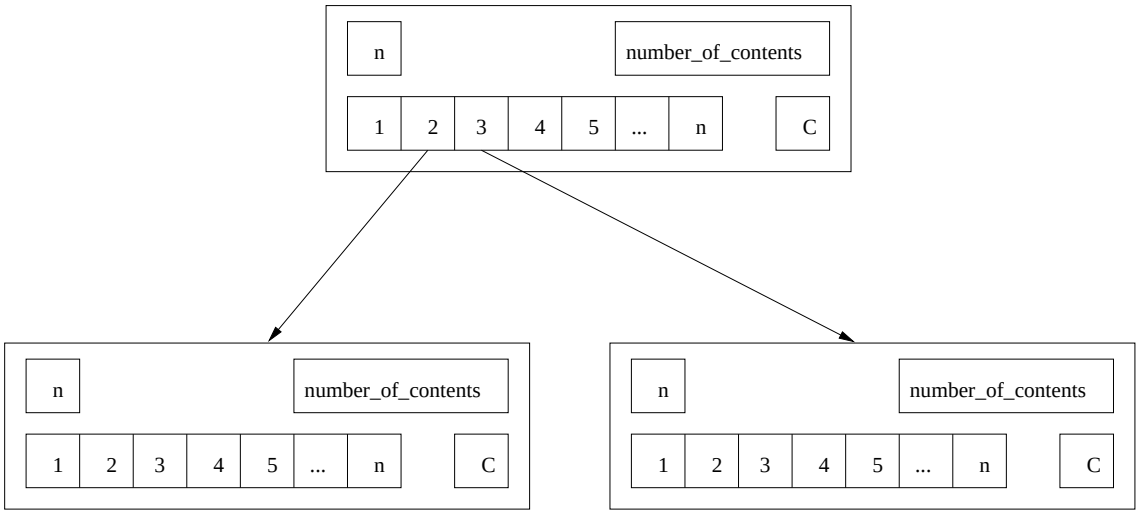

Figure 4: General structure of a trie

- A pointer to the `middle`.
- A pointer to the `right`.
- The `content` (C). This is a pointer to a structure which contains the elements belonging to a key.
- The `number` of contents (number_of_contents) stored in the subtree rooted at this node.

In Figure 5 the general structure of a ternary search tree is depicted. An example (with number_of_contents omitted) is depicted in Figure 3 (see Section 4.3.2).

B.1.3 Prefix trees with unique keys

The template class PTREE_UNIQUE is used whenever only contents with unique keys are to be stored. In this case two parameters have to be instantiated: the KEY, and the CONTENT.

The class KEY has to provide the following methods:

- *get_length()*: return the length of the key
- $[] \rightarrow int$: a function taking a position within the key as an argument and returning an index

The class CONTENT is used as a container for the content associated with a key.

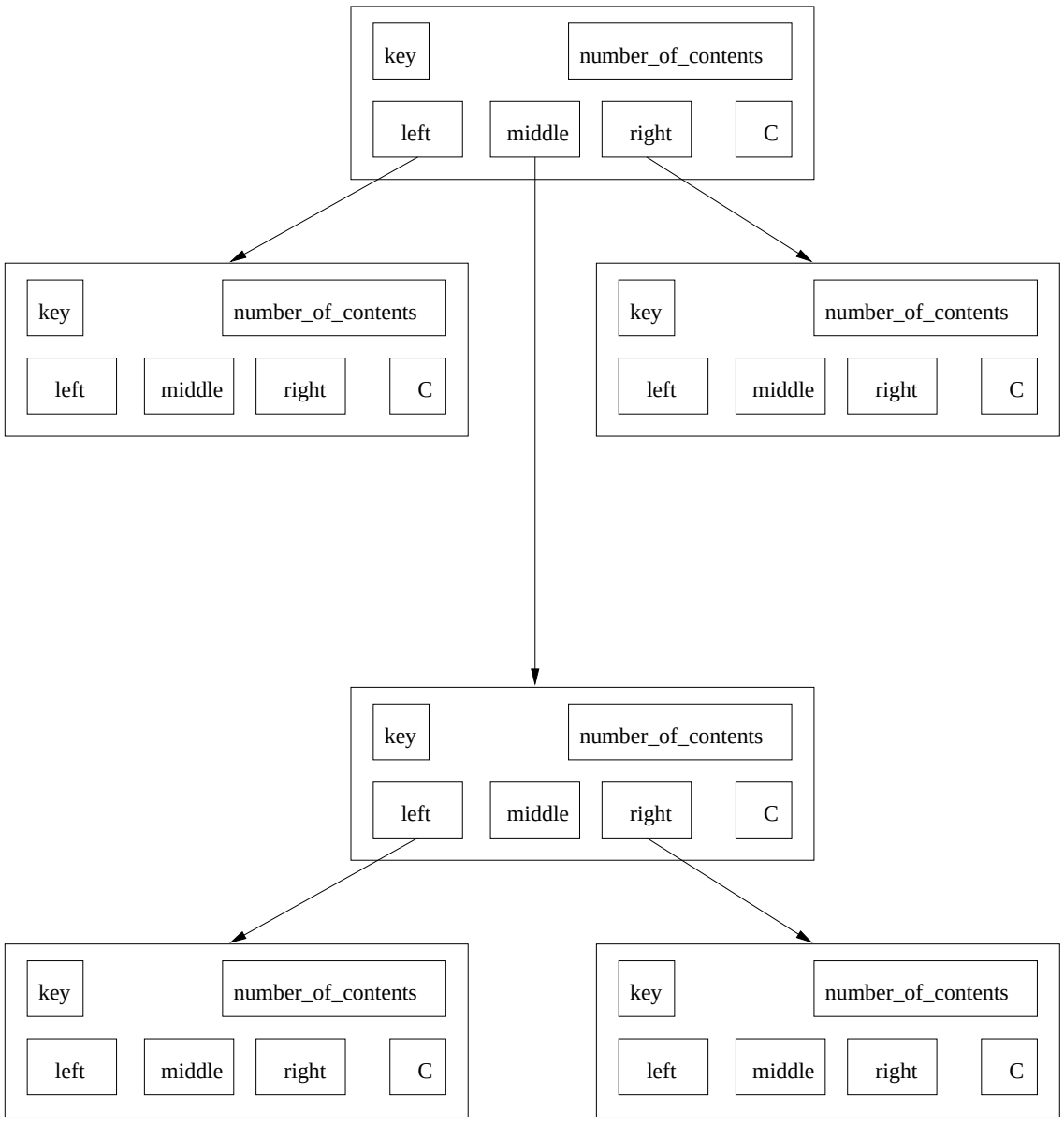

Figure 5: General structure of a ternary search tree

An example for the use of this class is the class RULES_TRIE described in Section B.2.8.

B.1.4 Prefix Trees with multiple keys

The template class PTREE is used whenever different contents with the same key are to be stored. In this case three parameters have to be instantiated: the KEY, the CONTENT_SET and the CONTENT_ELEMENT.

The class KEY is the same as for prefix trees with unique keys and the class CONTENT_ELEMENT is the same as the class CONTENT.

The class CONTENT_SET contains the different contents belonging to the same key. This class is necessary to allow multiple contents belonging to one key. It has to provide the following methods:

- *add(*CONTENT_ELEMENT*)*: adding one content
- *remove()* $\rightarrow$ CONTENT_ELEMENT: delete and return a content
- *get()* $\rightarrow$ CONTENT_ELEMENT: return a content
- a constructor which takes a content as parameter
- *is_empty()*: returns whether the content set is empty or not

An example for the use of this class is the class POLYSET_TRIE described in Section B.9.5.

B.2 Monoid

These classes represent the monoid. They are depicted in the class diagram 'monoid'. The two classes MONOID and TERM which form the link to the other classes of MRC V 1.0 are also depicted in the class diagram 'mrc'.

B.2.1 MONOID

This class is nothing else than a container for three other classes which form the presentation of a monoid. The data can be accessed directly thus avoiding a duplication of the methods and eliminating one indirection step. The only methods provided are a constructor and a destructor, as well as methods to read a monoid presentation from a file and to write it to a file.

The monoid consists of an ALPHABET, an ORDERing on the TERMs constructed from the ALPHABET and a set of RULES. Each of these concepts is encapsulated within a class.

B.2.2 CHARACTER

A character is given by its external `representation` and a `weight` required for the completion orderings, e.g. KBO, or representing a precedence on the characters. For performance reasons the weights are not used, but are stored with the orderings (see B.2.5).

B.2.3 ALPHABET

The alphabet is given by a list of characters. It is used for transforming representations from internal to external and vice versa. The position within this list is taken as the internal representation of a character. Thus the internal representation is an integer which can be matched to an external representation via an array-operation on the list. The match of an external to an internal representation is done by searching the list sequentially. This operation is not so important, because this transformation is normally only done at the beginning, when the input files are read and the data they contain is transformed into the internal data structures.

B.2.4 ORDER

The class ORDER is an abstract class which defines the common interface of all its subclasses. The task of an ordering is the comparison of two terms. The method *compare* has a special form which is required by LEDA, as it is used in connection with LEDA classes.

B.2.5 ORDER_LL

Up to now only one ordering is implemented namely the length-lexicographical ordering. For performance reasons, the weights of the characters are stored in this class which makes the lookup more efficient than function calls fetching the values from the class CHARACTER. It is obvious that this is not recommended in general. One advantage in this setting is, that the presentation of a monoid is fixed during one application. That is why the weights cannot change and therefore have not to be recomputed during runtime.

B.2.6 RULESET

This class represents a set of rules. A hybrid data structure is used to speed up operations. Here, too it is advantageous that the structure does not change after being initialized. The data structure consists of the two structures list of elements (`rule_list`) and prefix tree of elements (`rule_set`). The list of elements makes it possible to quickly enumerate the whole set of rules. The prefix tree on the other hand speeds up normalization (reduction of terms). The concept of prefix trees is explained in Section 4.3.

B.2.7 RULE

A RULE consists of two terms, the `left` hand side and the `right` hand side of the rule.

B.2.8 RULESET_Trie

This is an instantiation of the template class PTREE_UNIQUE with the parameters KEY $\rightarrow$ TERM (left hand side of a rule) and CONTENT $\rightarrow$ RULE (the rule itself). It is expected, that each key is unique for the set of rules. That is why an interreduced string rewrite system is requested.

B.2.9 RULESET_ITERATOR

This class is used to iterate over all elements of the associated `ruleset`.

B.3 TERM

A `term` (or word) is represented internally by an array of int. Each element of the array is an index which can be mapped to a CHARACTERs external representation using the `alphabet` of the `monoid`. Further the `length` of the term is stored.

The multiplication of two terms is realized as the concatenation of the two terms followed by computing the normal form with respect to the monoid presentation. Terms are not normalized during other operations especially not while reading the input.

B.4 TERMSET

This class is used to hold a set of terms (`termset`). These are the terms which arise from overlapping head terms of polynomials and left sides of rules. The data structure used to store the set is a list.

B.5 TERMSET_ITERATOR

This class is used to iterate over all terms of the associated `set` of terms.

B.6 COEFF

This class represents the `coeff`icients of the polynomials taken from $\mathbb{Q}[\mathcal{M}]$. It is used as an interface to the LEDA implementation of $\mathbb{Q}$. Thus it would be easy to replace the LEDA implementation by some other implementation of $\mathbb{Q}$ or any other field.

B.7 POLYNOM

Polynomials are represented as an ordered set of monomials with the TERM as key and the COEFFicient as the information. The head monomial is the maximal element of the set.

B.8 POLYNOM_ITERATOR

This class is used to iterate over all monomials of the associated `polynom`ial. In addition, the current monomial can be deleted.

B.9 Polyset

These classes are used to represent a set of polynomials. They are depicted in the class diagram 'polyset'.

B.9.1 POLYSET

This class describes a set of polynomials. A hybrid data structure is used consisting of a list of polynomials (`polynom_list`) and a prefix tree with the head terms as keys and the polynomial itself as information (`polynom_trie`). The prefix tree is used for efficient reduction whereas the list is necessary to allow a fair strategy when selecting polynomials from the set. Both data structures are consistent (representing the same set of polynomials) before and after any invocation of a public method.

In order to avoid duplicates a test is performed whether the polynomial which is inserted next can be reduced in one step to zero using a polynomial from the set. Those polynomials which can be reduced are not inserted. Besides duplicates also special multiples of polynomials already in the set are not inserted which is consistent with the theory.

B.9.2 POLYSET_CONTENT_SET

This class represents the set of all polynomials having the same head term. They are stored in a list in the same order as in the list `polynom_list` in POLYSET.

B.9.3 POLYSET_CONTENT_SET_ITERATOR

This class is used to iterate over all polynomials of the associated `polyset_content_set`.

B.9.4 POLYSET_CONTENT_ELEMENT

This class represents one polynomial. In addition to a pointer to the polynomial, a reference to the according element of the list `polynom_list` in the class POLYSET is stored.

B.9.5 POLYSET_Trie

This is an instantiation of the template class TRIE or TTREE with the parameters KEY → TERM, CONTENT_SET → POLYSET_CONTENT_SET and CONTENT_ELEMENT → POLYSET_CONTENT_ELEMENT.

B.10 CP

This class is representing a critical situation leading to an `spolynomial` which is computed and stored.

B.11 CP_SET

This class contains a list of critical pairs (see also B.10). Critical pairs are used in the procedure prefix.groebner_basis_of_right_ideal_1 described in Section 2 (see also B.12.3).

B.12 Ideal

These classes provide the procedures for prefix saturation and the Gröbner basis procedures as described in Section 2.1 and 2.2.

B.12.1 IDEAL

This class is the parent class of all other "IDEAL_" classes. It provides a procedure used by the other classes for normal form computation.

B.12.2 POLYSAT

This class contains no data and only two methods for the saturation of a polynomial and a set of polynomials. It is the parent class of all saturation classes which can be used together with the generic classes IDEAL_GENERIC and IDEAL_TWOSIDED_GENERIC described in Section B.12.3 and B.12.4 respectively.

B.12.3 IDEAL_GENERIC

This class implements the procedures prefix.groebner_basis_of_right_ideal_1 and prefix.groebner_basis_of_right_ideal_2 presented in Section 2.1. The `saturation` procedure associated can be the general one presented in Section 2.1 or one of the specialized procedures for plain or context-free group presentations as described in Section 2.5 and Section 2.6 respectively. There are two variants of the first procedure available: compute the Gröbner basis only and compute the Gröbner basis and interreduce it.

B.12.4 IDEAL_TWOSIDED_GENERIC

This class implements the procedure prefix.groebner_basis_of_two_sided_ideal presented in Section 2.1. The same saturation procedures can be associated as with IDEAL_GENERIC in Section B.12.3 whose methods are used to compute the Gröbner bases of right ideals.

B.12.5 POLYSAT_ENUMERATE

This class implements the saturation procedures described in Section 2.2. It is initialized with one polynomial of which the saturation set should be computed and stores the already computed subset of the prefix-saturated set and the set of polynomials still to be considered.

B.12.6 IDEAL_ENUMERATE

This class implements the Gröbner basis enumeration procedure described in Section 2.2. It stores the information representing the Gröbner basis which consists of the subset of the Gröbner basis of the `ideal` so far computed, the set of critical pairs (`B`) still pending, and a list of saturators (`polysat_list`) which are not yet finished.

It provides two methods: one to enumerate the Gröbner basis of a given set of polynomials generating an ideal, and one to decide whether the polynomials of a given set of polynomials lie in the ideal generated by a second set of polynomials. The latter is only a semi-decision procedure in the following sense. All polynomials which lie in the ideal will be identified as such as long as a fair strategy is used. Those not lying in the ideal will only be identified as such if the procedure terminates, else it will run forever, i.e. there will be no answer.

B.13 Group

These classes are used for monoids which in fact are groups. They are depicted in the class diagram 'monoid'.

B.13.1 GROUP

This class is derived from the class MONOID. It redefines several methods in order to allow additional operations on the alphabet and on the terms with respect to the additional information a group provides, namely the existence and representation of inverses.

B.13.2 ALPHABET_GROUP

This class is derived from the class ALPHABET. It has additional data, namely for each character of the alphabet it stores the index of the `inverse` character. It is supposed that each character of the alphabet has an inverse of length one that is an inverse character. Additional methods provided allow the access to these data.

B.13.3 TERM_GROUP

This class is derived from the class TERM. It provides additional methods for handling terms whose underlying monoid is a group, especially generation of the inverse term.

B.14 Fm

These classes provide the methods to compute prefix Gröbner bases in the free monoid ring as described in Section 2.3. No saturation is needed. The structure of the classes is depicted in the class diagram 'ideal'.

B.14.1 IDEAL_FM

This class is derived from the class IDEAL and provides an implementation of the procedure prefix.groebner_basis_of_right_ideal_fm.

B.14.2 IDEAL_TWOSIDED_FM

This class is derived from the class IDEAL_FM and provides an implementation of the procedure prefix.groebner_basis_of_two_sided_ideal_fm.

B.15 Fg

These classes provide the methods to compute prefix Gröbner bases in the free group ring as described in the Sections 2.4 and 2.7. The structure of the classes is depicted in the class diagrams 'monoid' and 'ideal'.

B.15.1 FG

This class is derived from the class GROUP and provides methods adapted to the free group presentation. Especially the rules are generated and not read from input.

B.15.2 POLYSAT_FG

This class provides the simplified saturation procedure available for the case of free group rings.

B.15.3 IDEAL_FG

This class is derived from the class IDEAL and provides an implementation of the procedure prefix.groebner_basis_of_right_ideal_fg.

B.15.4 IDEAL_TWOSIDED_FG

This class is derived from the class IDEAL_FG and provides an implementation of the procedure prefix.groebner_basis_of_two_sided_ideal_fg.

B.15.5 TODD_COXETER

This class is derived from the class IDEAL_TWOSIDED_FG and provides an implementation of the procedure extended_todd_coxeter_simulation described in Section 2.7.

B.15.6 TERMSET_SORTED

This class provides a set of terms (`termset`) which is sorted and has no duplicates.

B.16 Plain_group

These classes provide the methods to compute prefix Gröbner bases in the plain group ring as described in Section 2.5. The structure of the classes is depicted in the class diagrams 'monoid' and 'ideal'. The class POLYSAT_PG is used as saturator for the procedures of the classes IDEAL_GENERIC and IDEAL_TWOSIDED_GENERIC as described in the Sections B.12.3 and B.12.4 respectively.

B.16.1 PLAIN_GROUP

This class is derived from the class GROUP and provides the methods adapted to the plain group presentation.

B.16.2 POLYSAT_PG

This class is derived from the class POLYSAT and provides specialized procedures for the saturation of polynomials and sets of polynomials as described in Section 2.5.

B.17 Cf_group

These classes provide the methods to compute prefix Gröbner bases in the context-free group ring as described in Section 2.6. The structure of the classes is depicted in the class diagrams 'monoid' and 'ideal'. The class POLYSAT_CFG is used as saturator for the procedures of the classes IDEAL_GENERIC and IDEAL_TWOSIDED_GENERIC as described in the Sections B.12.3 and B.12.4 respectively.

B.17.1 CF_GROUP

This class is derived from the class MONOID and provides the methods adapted to the context-free group presentation. The order of the characters is: characters of the finite part of the presentation and then characters of the free group part of the presentation. Thus it is sufficient to store the free group part in `fg` and the number of finite elements in `no_finite_elements`. The alphabet contains `no_finite_elements` characters belonging to the finite group, the rest belongs to the free group. The free group part is used for all computations concerning subterms belonging to the free group part only.

B.17.2 POLYSAT_CFG

This class is derived from the class POLYSAT and provides specialized procedures for the saturation of polynomials and sets of polynomials as described in Section 2.6.

List of Procedures

List of Figures