How to compute an intersection

Carlo Traverso
Dipartimento di Matematica
Università di Pisa
traverso@dm.unipi.it

Massimo Caboara
Dipartimento di Matematica
Università di Pisa
caboara@dm.unipi.it

Abstract

It is "well known" that there are two algorithms to compute the intersection of two ideals $I := (f_1, \ldots, f_n)$, $J := (g_1, \ldots, g_m)$ of a polynomial ring (or of two submodules of a free module); one of them (the "tag variable" algorithm, [Sp], [GTZ]) eliminates t from $tI + (1+t)J$, where t is a new variable (instead of $tI + (1+t)J$ one can use $tI + (1-t)J$, this form being more usual but algorithmically just the same); the other, (the "syzygy" algorithm, [Wa]) computes the module of first syzygies of $(f_1, \ldots, f_n, g_1, \ldots, g_m)$, and evaluates each syzygy in $(f_1, \ldots, f_n, 0, \ldots, 0)$. The computation of syzygies is a subproduct of the computation of a Gröbner basis of $I + J$.

The discussion on the superiority of one method with respect to the other has not, up to now, been settled. The usual argument in favor of the syzygy algorithm is the following: computing a Gröbner basis of $I + J$ without conditions on the ordering is easier than computing a Gröbner basis of $tI + (1+t)J$ with respect to an elimination ordering, at least in some special cases, (the term-ordering is worse, and we have one variable more). This argument is however not conclusive, since the syzygy algorithm gives a basis which is not Gröbner, hence often one has to perform another Buchberger algorithm, since usually the intersection is only an intermediate result. The tag variable algorithm, that gives directly a Gröbner result, might at the end be superior.

In this paper we settle the issue, showing that there is a form of the tag variable intersection algorithm that consists precisely of performing the syzygy algorithm in an improved form and then a Buchberger algorithm on the result.

To explain this equivalence, we have to give some details on the algorithms, in particular on the Buchberger algorithm for modules.

Some other algorithms have been proposed to compute multiple intersections, and to compute ideal and module transporters $M : N$, $M : J$, (M, N submodules of a free A-module, J an ideal of A), apparently avoiding the use of an intersection algorithm. We show that these algorithms too fall in the same frame.

We show moreover that the improved tag variable algorithm, if the two ideals are given through Gröbner bases and are homogeneous, allows the use of the Hilbert-driven Buchberger algorithm, see [Tr], hence a considerable speed-up can be obtained. This feature can be used also for non-homogeneous ideals, homogenizing the input and de-homogenizing the output. We expect this to be the best performing algorithm for intersection. *The implementation is not finished (an easy variation of the Hilbert function algorithm is needed) and we expect to report the timings in the final version of the paper.*

Several algorithms have been proposed for multiple intersection or transporter; we show that these too can be interpreted in the same framework; however their performance is doubtful, at least in the "hard" cases, since the Hilbert function cannot be used.

1 Notations, the Buchberger algorithm

We give a short description of the Buchberger algorithm. This description is mainly to fix notations and to put some light into a few points that we have later to modify. The reader is assumed to know Gröbner basis and Buchberger algorithm, see e.g. [Bu], [CLS], [BW], [Eis].

The Buchberger algorithm, in its ground form, is concerned with ideals of a polynomial ring $k[X]$, $X = (x_1, \ldots, x_n)$ over a field k, or more precisely on finite sets of polynomials seen as generators of an ideal, and computes a Gröbner basis of this ideal (we do not need the definition of Gröbner basis). Polynomials are seen as sum of monomials, composed of a coefficient $c \in k$ and a power-product X^I, a product of powers of indeterminates. The set of power products is a multiplicative monoid, isomorphic to the additive monoid N^n.

We assume to have a total ordering on the set of power products, that is compatible with the monoid operations and has the power product 1 (i.e. $0 \in N^n$) as minimal element. Such an ordering is called a term-ordering, and polynomials are represented sorting the monomials in descending term-ordering. The leading power product $Lpp(f)$, leading monomial $Lm(f)$ and leading coefficient $Lc(f)$ of a polynomials are precisely what you imagine from the name.

If (f, g) is a polynomial pair, define $Lpp(f, g)$ the least common multiple of $Lpp(f)$ and $Lpp(g)$, and $Sp(f, g) = \frac{Lc(g)Lpp(f,g)}{Lpp(f)} f - \frac{Lc(g)Lpp(f,g)}{Lpp(g)} g$. Remark that if $Lpp(g)$ divides $Lpp(f)$, then $Lpp(Sp(f, g)) < Lpp(f)$. Our definition of Sp includes both the usual Sp and the rewriting of f by g.

The Buchberger algorithm consists in adding elements of the ideal to the initial set of generators; we denote by G this increasing set of generators. After an initialization phase, the algorithm is governed by a set of pairs of G, the pair queue. When the queue is empty, G is a Gröbner basis, and after an optional final normalization the algorithm is terminated.

The body of the algorithm consists in repeatedly choosing a pair (g_i, g_j) out of the pair queue (the choice is irrelevant for the correctness of the algorithm, but has big influence on the total execution time; a specification of the procedure to choose a pair is called a selection strategy). Then one computes $f = Sp(g_i, g_j)$, and this step of the algorithm is a division (or simplification) loop: one repeatedly chooses a g_r such that $Lpp(f)$ is multiple of $Lpp(g_r)$ (if any), and puts $f = Sp(f, g_r)$. The end of the loop is when no g_r such that $Lpp(g_r)$ divides $Lpp(f)$ is found. Whenever many $Lpp(g_r)$ divide $Lpp(f)$, one may choose between them: the *simplification strategy* is used to choose.

At this point, if $f = 0$ one continues the outer loop choosing another pair; otherwise one adds f to the list G, adjusts the pair queue, adding some pairs (f, g_i) and optionally deleting some of the existing pairs, following one of the possible strategies (*Useless pair elimination*, see [Bu], [GM]) and continues the outer loop choosing another pair.

2 Intersections

Here we describe the two classical algorithms to compute submodule intersections, together with some less known variants.

The tag variable algorithm: The algorithm relies on the following

Theorem 1 *Let N, H be submodules of a free A-module M, $A := k[X]$. Let t be a new indeterminate, and consider the $A[t]$ modules M_t, N_t, H_t, $M_t := M \otimes_A A[t]$, etc.; a presentation of N, H lifts to a presentation of N_t, H_t. Then $(tN_t + (1+t)H_t) \cap M = N \cap H$.*

Proof. An (incomplete) proof is the following: consider $\gamma = t\alpha + (1+t)\beta$, and assume $\gamma \in M$, $\alpha \in N$, $\beta \in H$. Since $\gamma = \beta + t(\alpha + \beta)$, and t does not appear in γ, this means that $\alpha + \beta = 0$ (hence $\alpha = -\beta \in H$) and $\gamma = \beta = -\alpha \in N \cap H$. The converse inclusion is straightforward. The proof is incomplete, since we cannot a priori assume $\alpha \in N$, but only $\alpha \in N_t$. The proof can anyway be adjusted (induction on the degree of t in α; the proof above is the level 0 of induction). The present "proof" proves something else that will be revealed in a next section. Q.E.D.

The algorithm is hence a special case of elimination: this can be solved with a Buchberger algorithm with respect to an elimination term-ordering (see [Sp], [GTZ], etc.).

The other algorithm is based on the following:

Lemma 2 *Let $A^r \to N \to 0$, $A^s \to H \to 0$ be exact; hence the induced map $A^{r+s} \to N+H \to 0$ is exact, and let $A^l \to^{\phi} A^{r+s} \to^{\psi} N + H \to 0$ be an exact extension. Let $\pi : A^{r+s} \to A^r$, $\iota : A^r \to A^{r+s}$ be the canonical projection and injection. Then $\psi\iota\pi\phi(A^l) = N \cap H$.*

Proof. The proof is almost immediate, see [Eis]. To find explicitly these relations, see [Sch], [MM], [BW], or a following section. Q.E.D.

Hence to find the intersection $N \cap H$ one can find the relations between the generators of N and H (i. e. the map ϕ).

In [Eis], Exercise 15.42, a third algorithm is reported (see also [BGS] if you can): compute the first coordinate of the kernel of the map defined by the matrix

$$\begin{pmatrix} 1 & f_1 & \dots & f_s & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 & g_1 & \dots & g_t \end{pmatrix}$$

where $(f_i) = I$ and $(g_j) = J$. The proof of the correctness of the algorithm is immediate. The algorithm allows obvious extensions to the case of submodules of a free module M (the map being in that case a map to $M \oplus M$, and the initial 1 of each line being interpreted as a square matrix).

These algorithms have variants for multiple intersections. Assume that we want to compute $I_1 \cap \cdots \cap I_n$. The tag variable algorithm adds variables $t_1, \dots, t_n$, with the relation $t_1 + \cdots + t_n + 1$, and eliminates the t_i from $t_1 I_1 + \cdots + t_n I_n$. The third algorithm adds further lines with the generators of the other ideals.

3 The Buchberger algorithm for modules

Let $M := A^m$ be a free A-module, $A := k[X]$ a polynomial ring, $X := (x_1, \dots, x_n)$. A power product in M is a Te_j, where T is a power product in $k[X]$, and $e_1, \dots, e_m$ is the canonical basis of A^m, $e_i := (0, \dots, 0, 1, 0, \dots, 0)$, with 1 at the i-th place.

A term-ordering on M, compatible with a given term-ordering of A, is a total ordering on the set of power products, compatible with the A-module product; more precisely, we require that if $T_1 \geq T_2$ and $S_1 \geq S_2$, T_i, S_j power products in A, M respectively, we have $T_1 S_1 \geq T_2 S_2$.

The definition of Gröbner basis of a submodule of M, and the Buchberger algorithm to compute it, are obvious generalization of the corresponding concepts for ideals. See [Eis], [BW].

The obvious representation of an element of M as n-tuple of elements of $k[X]$ is not convenient for the Buchberger algorithm unless the term-ordering is "lexicographic on the components", i. e. if $Te_i > Se_j$ if $i < j$. Only in this case, if the polynomials are represented as usual as sum of monomials in decreasing order, the leading power product is the first non zero power product. A better representation gives an element $m \in M$ as sum of monomials $\sum c_r T_r e_{i_r}$, sorted in decreasing term-ordering. Hence, an element of M can be seen as a polynomial in $k[X, e_i]$, linear homogeneous in the e_i.

Using this idea, we want to show that:

a. any implementation of the Buchberger algorithm for ideals can be used to compute Gröbner bases for modules.

b. any implementation of the Buchberger algorithm for ideals with a small modification gives an efficient implementation of the Buchberger algorithm for modules.

The symmetric algebra $S(M) := \bigoplus_{i=0}^{\infty} M^{\otimes i}$ is isomorphic to a polynomial ring in $n + m$ indeterminates:

the X and a new set of variables E_i corresponding to the canonical basis e_j of M. Submodules N of M correspond both to ideals of $I \subseteq A$ generated by elements homogeneous of degree 1 in the e_i, $I = \bigoplus_{i=1}^{\infty} N^{\otimes i}$, or to ideals J such that $\bigoplus_{i=1}^{\infty} M^{\otimes i} \subseteq J \subseteq \bigoplus_{i=2}^{\infty} M^{\otimes i}$. We have $J = N \oplus \bigoplus_{i=2}^{\infty} M^{\otimes i}$.

To compute a Gröbner basis of a submodule N generated by $n_i = \sum_j n_i j e_j$, we can either:

a. compute the Gröbner basis of the ideal generated by $\bar{n}_i = \sum n_i j E_j$, and take all elements of degree 1 in the E_i.

b. compute the Gröbner basis of the ideal generated by the $\bar{n}_i$ and the $E_i E_j$.

c. the same as a), arranging the critical pairs in ascending degree in the E_j of their associated power product (the power product associated to a pair is the least common multiple of the leading power products of the two elements of the pair), and interrupting the computation after considering all critical pairs of degree 1.

d. the same as a), discarding systematically all critical pairs (g_i, g_j) such that $Lpp(g_i)$ and $Lpp(g_j)$ (that are of degree 1 in the E_r) contain a different E_r.

The four possibilities are shown in increasing efficiency: the first one makes a lot of computations to find elements of a Gröbner basis that are later discarded, the second at the end reduces rapidly to 0 all the pairs that in c) are discarded at the end, and in d) are discarded at once.

It is not difficult moreover to show that every term-ordering on a free module M over a polynomail ring $k[X]$, which is compatible with a fixed ordering of $k[X]$ and gives the same ordering on all the components, can be extended (not uniquely) to a term-ordering of $S(M)$, see [CS]. For more general module term-orderings, see [RR].

Another approach can be useful for $N \subseteq M$, M a sub-$k[X]$-module of a polynomial ring $k[X,Y]$ generated by a finite set of monomials in the Y. It consists in the Buchberger algorithm in $k[X,Y]$ with the following three differences:

1. A critical pair (g_i, g_j) is considered only if the following condition is satisfied: let $Lpp(g_i) = X^I Y^J$, $Lpp(g_j) = X^H Y^K$. Then $J = K$.

2. When simplifying a polynomial f, consider only g_j such that $Lpp(f) = X^I Y^J$ and $Lpp(g_j) = X^L Y^J$, with the same Y^J.

3. When discarding useless pairs, do not discard (g_i, g_j) if $Lpp(g_i)$ and $Lpp(g_j)$ are coprime.

(An example of 3): $(x_1^2 + y, x_2) \subset k[x_1, x_2] \oplus yk[x_1, x_2]$ is not a Gröbner basis, the addition of $x_2 y$ is needed since $x_2 y$ is not $k[X]$-multiple of x_2. With the notation that uses a tag variable for every component, $(x_1^2 + y, x_2)$ is written $(x_1^2 e_1 + e_2, x_2 e_1)$, hence the pair cannot be discarded.)

This last version has the advantage that one has to consider fewer variables (only the Y, and not one for each monomial in the Y), and is especially useful when an overring is considered sometimes as a module (typically, computing in integral extensions, [CT]). The three conditions above are a faithful translation of d) above, if one substitutes the generating monomials (including 1) with different variables E_i. The Y are called in this context "module variables". The idea is that a power product $X^H Y^K$ divides $X^R Y^S$ only if $K = S$ and X^H divides X^R (in the usual sense) since we are allowed to multiply by the X but not by the Y. There are several cases in which an ordering of this kind is very useful: we will see some in this paper, and more are contained in [Ca1].

The three modifications of the Buchberger algorithm above can be justified in the following context. Consider $M := k[X,Y]$ as R-module and $R := k[X]$. Forget the ring operation in M and consider divisibility, GCD and lcm in the sense of modules:

1. given elements $a, b \in M$, a divides b iff exists $c \in R$, $ca = b$;

2. a common multiple of a and b may not exist; hence the S-polynomial of two elements of M may not exist, and the corresponding critical pairs can be discarded;

3. the GCD of a and b in M not only may not exist, but it is never 1, the multiplicative identity, that does not exist in M. The corresponding criterion that allows to discard pairs in the ring case is never verified in the module case.

Hence, the difference is not in the Buchberger algorithm, but in the definition of the operations on the power products. This is hence

Summarizing, elements of a free $k[X]$-module can be seen alternatively as vectors or as polynomials in an extension $k[X,Y]$; a list of generators of a submodule can be seen either as a matrix (whose columns are the generators represented as vectors) or as lists of polynomials of $k[X,Y]$, obtained multiplying to the left the matrix above by a row vector of (different) power products in Y.

If we are interested in homogeneous modules, we have to specify for each row of the matrix (or equivalently for each power product in Y that comes in the representation) a suitable weight. This might be induced by a weight on the variables Y, but sometimes it is more useful to consider it a componentwise offset, or a sum of a weight and an offset.

(How to compute Hilbert functions for modules from Hilbert function for ideals)[BCRT], [BS]

4 Syzygies

It is known, [MM], [GM], [Sch] that if $G := \{g_1, \ldots, g_r\}$ is a Gröbner basis of a submodule N of a free A-module $M \cong A^m$, $A := k[X]$, $N := Im(\phi)$, $\phi : A^r \to A^m$, then a set of generators of the module K of relations between the g_i, $0 \to K \to A^r \to A^m$ can be computed from a suitable set Γ of pairs (g_i, g_j). This set can be determined as follows.

Let $T_i e_{n_i} = Lpp(g_i)$, and f_i the canonical basis of A^r. Consider the map $\phi : A^r \to A^m$, $\phi(f_i) = T_i e_{n_i}$.

Then $\ker\phi$ is generated by the so-called Taylor basis Γ', indexed by all pairs (i,j), $\gamma_{i,j} = S_{ij}f_i - S_{ji}f_j$, $S_{ij} = T_j/GCD(T_i,T_j)$.

A subset of Γ' is sufficient to generate K; practical ways to determine such a subset are given in [Bu] and [GM]; let Γ be the set of pairs indexing such a subset. The elements of Γ' not in Γ correspond to the useless pairs in the Buchberger algorithm.

The Buchberger algorithm applied to G (as described in [Bu] and [GM]) checks that for every pair $(g_j, g_j) \in \Gamma$, $Sp(g_i, g_j) \Rightarrow 0$, i.e. we have $Sp(g_i,g_j) = \sum c_r S_r g_{i_r}$, for suitable constants c_r and power products S_r, and the sequence $S_r T_{i_r}$ is strictly decreasing. Then $\gamma' = \gamma_{i,j} - \sum c_r S_r f_{i_r}$ is a relation between the g_l. It is proved in [MM] that the γ' generate the module of relations.

(Remark that another criterion for useless pairs is known, namely a pair (g_i, g_j) is useless if $Lpp(g_i)$, $Lpp(g_j)$ are coprime; however these pairs are necessary while computing syzygies, even if the corresponding syzygy is easy, consisting of $g_i g_j - g_j g_i{=}0$).

If G is not a Gröbner basis, one can find a (redundant) Gröbner basis $G' \supseteq G$ with the Buchberger algorithm. As a subproduct of the Buchberger algorithm one finds the relations between the G': if a pair simplifies to 0, then it gives a relation as above; if it does not, then adding the result of the simplification to G' one restores the fact that pair simplifies to 0.

From a set of generators of the relations between the G', one recovers a set of generators of the relations between the original G by back substitution: whenever we add an element to G', we add to the relations one stating that the new element is linear combination of the preceding ones, so relations between the G are obtained easily.

The idea is very similar to obtaining the Bézout identity from the euclidean algorithm; hence, a better algorithm can mimick the extended euclidean algorithm. Namely, instead of considering elements of M, we consider pairs consisting of an element of M and a vector of A^r (r is the cardinality of G). We start with (g_i, v_i), v_i being the i-th member of the canonical basis, $v_i := (0, \ldots, 1, \ldots, 0)$.

Whenever in the Buchberger algorithm we perform a computation on the first component, we perform the same operation on the second component. In this way, any pair (g, v) that we compute satisfies the assertion $g = \langle v, G\rangle$, where $\langle \_, \_ \rangle$ denotes the canonical extension of the scalar product $A^r \times M^r \to M$. Whenever in the Buchberger algorithm we obtain a zero (as first element of the pair) we have obtained a relation between the G_i. One sees easily that the result is precisely the same that one would have obtained with the procedure sketched above.

We remark now that this procedure (we can call it "extended Buchberger", copying from the extended Euclid algorithm), can be implemented as an interrupted Buchberger algorithm for modules: instead of M we consider $\bar{M} := M \oplus A^r$, and instead of g_i we consider $g_i + e_i$ (e_i elements of the canonical basis of A^r). More explicitly, we consider the matrix

$$\begin{pmatrix} g_1 & \ldots & g_n \\ 1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 1 \end{pmatrix}$$

in which the g_i might themselves be (column) vectors.

We choose a term-ordering such that power products of M are larger than power products of A^r. The term-ordering on A^r is arbitrary, but it is useful to use a term-ordering such that $T_i e_i > T_j e_j$ if $T_i Lpp(g_i) > T_j Lpp(g_j)$. Now consider the Buchberger algorithm for $\bar{M}$ discarding the critical pairs between elements with Lpp in A^m. The resulting elements with Lpp in A^r (hence in A^r) generate the relations between the g_i. This Buchberger algorithm for $M + A^r$ is substantially identical to the extended Buchberger algorithm for M.

If we do not discard the pairs with Lpp in A^r, the computation gives a Gröbner basis of the relation module. Hence an implementation of Buchberger algorithm for modules can be used, without modifications, to compute syzygies.

5 Intersections

Syzygies can be used to compute intersections. Let $H := (h_i)$ and $N := (n_j)$ be submodules of a free module M over $A := k[X]$. The syzygy algorithm to compute the intersection between H and L computes the relations between the (h_i, n_j) and then for each generator γ of the module of relations takes the scalar product $\langle \gamma, (h_i, 0)\rangle$.

Assume that the syzygies are only used to compute the intersection, and then discarded. We show how to minimize the computations and the storage.

To avoid the final computation, one can simply:

a. avoid considering the components of the syzygies that correspond to the n_j, since they are anyway discarded at the end.

b. perform at the very start the scalar product that is needed at the end. This consists in applying the distributive property of the scalar product.

(Item a. is obviously advantageous, since we avoid to perform computations whose result is at the end multiplied by zero; the convenience of b. might be questioned —we consider a polynomial instead of a vector of polynomials, but the single polynomial might be more complicated that the vector— but in practice this is not the case.)

In matrix form, the original algorithm would compute the Gröbner basis of the matrix

$$\begin{pmatrix} f_1 & \ldots & f_s & g_1 & \ldots & g_r \\ 1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & \ldots & 1 \end{pmatrix}$$

while after improvement a) we have to compute

$$\begin{pmatrix} f_1 & \dots & f_s & g_1 & \dots & g_r \\ 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix}$$

and after improvement b. we compute just the two-rows matrix

$$\begin{pmatrix} f_1 & \dots & f_s & g_1 & \dots & g_r \\ f_1 & \dots & f_s & 0 & \dots & 0 \end{pmatrix}$$

and, as suggested, we can instead represent the matrix adding a tag variable t and multiplying to the left by $(t, 1)$. The variable t should be eliminated to compute the intersection.

Wait a moment: we have already seen that! In fact, what we am computing is just a Gröbner basis of $(t+1)f_1, \dots, (t+1)f_s, tg_1, \dots, g_r$, i.e. of $(t+1)H + tN \subseteq tk[X] \oplus k[X]$.

The usual tag variable intersection algorithm is the same thing, but t is considered as ring variable.

But indeed, looking closely at the tag variable intersection algorithm, I can improve it: if the tag variable t is considered to be a "module variable", in the sense of a preceding section, the algorithm still computes the intersection (the incomplete proof given in a previous section now is correct!) and the algorithm is simpler: I do not have to consider pairs (f', f'') such that $Lpp(f')$ contains t and $Lpp(f'')$ doesn't. Indeed, these pairs can easily proved to reduce to 0 (even if the standard criteria cannot prove it).

So, we have reconciled the two algorithms. Both can be improved to become coincident. Indeed, in the tag variable algorithm, if we are not interested to have an answer that is Gröbner, we may discard all pairs of elements that do not contain a t. And in the syzygy algorithm, if we want to have an answer that is Gröbner, we have to perform a Buchberger algorithm on the result.

There is still a difference between the two, in the timings of the different phases: if we first completely compute syzygies, and then a Gröbner basis of the intersection, this can be seen as interrupting the simplification phase of the Buchberger algorithm and resuming it later. Whether this is advantageous or not is not yet clear, and further experiments might be needed to settle the issue.

The syzygy algorithm can be more convenient if we do not need an answer that is a Gröbner basis; for example, if we compute an intersection of two ideals, and we intersect the result with the third, then we do not really need a Gröbner basis of the first intersection. However, the result is usually more complicated than a Gröbner basis, hence the subsequent computation is more difficult.

If instead we start from two ideals of which we have a Gröbner basis, the tag variable algorithm allows some improvements, allowing to discard several critical pairs. This will be discussed in the following sections.

Just a few comments on the "third algorithm". We have to compute the kernel of a map, i.e. the syzygies of the columns of a matrix, then take the first component of those syzygies. We can hence avoid to compute these components, as we have proposed for the first improvement of the syzygy algorithm. Hence we have to compute the Gröbner basis of the module generated by the columns of the matrix

$$\begin{pmatrix} 1 & f_1 & \dots & f_s & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 & g_1 & \dots & g_t \\ 1 & 0 & \dots & 0 & 0 & \dots & 0 \end{pmatrix}$$

To compute the module Gröbner basis, we multiply to the left by $(t, s, 1)$, t, s being module variables; then eliminate t and s. The first column is $t+s+1$, the others are generators of tI and sJ respectively; this means that we are performing the multiple intersection tag-variable algorithm.

5.1 Useless pair avoidance

Assume that we have to compute the intersection of two ideals I, J of which we have a Gröbner basis, $F := \{f_i\}$ $G := \{g_j\}$ respectively.

When computing a Gröbner basis with input $H := tF \cup (t+1)G$ we have to consider the critical pairs of elements of H. But we can remark that the pairs (tf_i, tf_h) will surely be simplified to 0 by elements of tF, exactly in the same way as (f_i, f_h) is simplified to 0 by elements of F. In the same way the pairs $((t+1)g_j, (t+1)g_h)$ will be simplified to 0 by elements of $(t+1)G$.

An optimized implementation of intersection can take advantage of this remark. The same remark can be useful for the syzygy algorithm, if we know the syzygies of F and G; but apparently this is more difficult and of less use.

Using either algorithm, if we don't need a Gröbner basis of the intersection module, we may discard a pair if the leading power products of both its element that don't contain the variable t.

We show now that, although it seems completely equivalent to the tag variable algorithm, the "third algorithm" does not allow the aforementioned optimization.

Consider the module generated freely by $t, s, 1$ (with an ordering eliminating t and s, $t > s$) and the submodule M generated by $tf_i, sg_j, t + s + 1$; to compute a Gröbner basis of M we can avoid all the critical pairs between elements of tf_i, and all the critical pairs between the sg_j; these elements moreover do not build critical pairs since they refer to different components; we have to start computing the critical pairs between the tf_i and $t + s + 1$. Each of these pairs generates an element $-(s + 1)f_i$, hence we are again to the standard form of the tag-variable module algorithm. But in this preliminary seemingly innocuous step we have lost the information that the pairs $((s+1)f_i, (s+1)f_j)$ go to 0, i.e. the information that we start from a Gröbner basis of I. If we start from a set of generators that are not a Gröbner basis the drawback is similar, since we first compute a Gröbner basis of tI just to discard the knowledge that it is a Gröbner basis recomputing it divided by t and multiplied by $(s + 1)$ instead. In the discussion below, we use the notation $GB(I, \tau)$ for the reduced Gröbner basis of an ideal I wuth respect to a term-ordering τ.

Proposition 3 *Let $R := k[x_1, \dots, x_n]$ and let $M, N \subset R^r$ be R modules. Then, using the matrix notation for the modules, the Gröbner basis of* $\begin{bmatrix} M & N \\ \mathbf{0} & N \end{bmatrix}$ *with respect to the module term-ordering μ is*

$$\begin{bmatrix} \mathrm{GB}(M+N, \mu_{|R^r}) & \mathbf{0} \\ \mathbf{F} & \mathrm{GB}(M \cap N, \mu_{|R^r}) \end{bmatrix}$$

if μ is a block ordering on $R^r \oplus R^r$ for which the first component is the biggest.

Proposition 4 *If I is an ideal of $R := k[x_1, \dots, x_n]$, t is a new variable and τ is an elimination ordering for t on $R[t]$, then $GB((I, (1+t)I), \tau) = GB(I, \tau_{|R})$. If t is a module variable, then*

$$GB((I, (1+t)J), \tau) = \begin{bmatrix} GB(I+J, \tau_{|R}) & 0 \\ F & GB(I \cap J, \tau_{|R}) \end{bmatrix}$$

Note that, using the tag variable module algorithm, the computation of $M \cap N$ has as side effect the computation of a Gröbner basis of $M+N$.

5.2 Hilbert-driven Intersection

If the modules (ideals) M, N are homogeneous with respect to a system of weights $\mathbf{w}$, then the tag variable intersection algorithms may be easily adapted to keep the homogeneity. The tag variables had to be given a suitable weight. We denote the Hilbert-Poincaré series of M w.r.t. a system of weights $\mathbf{w}$ as $\mathbf{H}^{\mathbf{w}}_M$, and its numerator in unsimplified form as $\langle M \rangle$.

Proposition 5 *Let $R := k[x_1, \dots, x_n]$ and $I, J \subset R$ be $\mathbf{w}$-graded ideals. Let t_1, t_2 be new variables. Then*

1. *the ideal $(t_1 I, (t_1+t_2)J) \subset R[t_1, t_2]$ is graded with respect to the system of weights*

$$\mathbf{u} := \left[\begin{array}{c|cc} & 1 & 1 \\ [\mathbf{w}] & \vdots & \vdots \\ & 1 & 1 \end{array} \right]$$

2. *Let τ be a term-ordering on $R[t_1, t_2]$ such that*
   a. *it is an elimination ordering for t_1.*
   b. *the variable t_2 is block-lesser than the other variables.*

   Then $(GB(t_1 I, (t_1+t_2)J, \tau))_{|t_2=1} \cap R = I \cap J$.

Since $Lpp_\sigma(t_1 I, (t_1+t_2)J) = t_1 Lpp_{\mu_{|R}}(I) \cup t_2 Lpp_{\mu_{|R}}(J)$ for suitable orderings σ and μ, the Hilbert function of $(t_1 I, (t_1+t_2)J)$ is quite easy to compute.

Proposition 6 *Let $R := k[x_1, \dots, x_n]$ and $M, N \subset R^r$ be $\mathbf{w}$-graded R modules. Let t_1, t_2 be new module variables and τ a module term-ordering on $t_1 R^r \oplus t_2 R^r$ fulfilling conditions of proposition 5, 2. Then*

1. *The module $(t_1 M, (t_1+t_2)N) \subset t_1 R^r \oplus t_2 R^r$ is graded with respect to the system of weights*

$$\mathbf{u} := \left[\begin{array}{c|cc} & 1 & 1 \\ [\mathbf{w}] & \vdots & \vdots \\ & 1 & 1 \end{array} \right]$$

2. *$GB(t_1 M + (t_1+t_2)N, \tau) \cap R^r = GB(M \cap N, \tau_{|R^r})$.*
3. *$\mathbf{H}^{\mathbf{u}}_{(t_1 M + (t_1+t_2)N)} = \mathbf{H}^{\mathbf{w}}_{M+N} + \mathbf{H}^{\mathbf{w}}_{M \cap N} = \mathbf{H}^{\mathbf{w}}_{M \oplus N}$.*

Proposition 6 allows us to use the Hilbert-Driven tecniques [Tr], [CDR] either for the module or ring tag variable intersection algorithms. The use of module variables for the ideal intersection may be useful, see the examples in section 5.4. If we happen to know the Hilbert function of $M+N$ or $M \cap N$ we can skip even more pairs, see [Ca2].

If M or N are not homogeneous we may homogenize them. From the discussion about pair avoidance, it is clear that we are only anticipating some of the computations.

5.3 Multiple intersections

If we have to intersect the modules $N_1, \dots, N_r \subseteq M$, we can add tag variables t_i, and eliminate them from the module

$$\sum_i t_i N_i + (1 + \sum_i t_i) R$$

The t_i are either ring or module variables, the second possibility being more efficient.

Another algorithm can be made as follows: the intersection of many submodules $M_1, \dots, M_r$ of M can be seen as the intersection of $\Delta \subseteq M^r$ with $M_1 \oplus \dots \oplus M_r$, where Δ, the diagonal, is the image of the map

$$\begin{array}{ccc} M & \to & M^r \\ m & \to & (m, \dots, m) \end{array}$$

We can compute the intersection with one of the algorithms for intersection of two submodules. The two algorithms sketched here substantially coincide. If it is more efficient using the algorithms described above or repeatedly apply the usual intersection algorithm depends on the input, but as a general rule, the repeated application of the Hilbert-driven algorithm is the winning choice.

Remark 7 *Since $M_1 \cap \dots \cap M_i$ grows smaller and smaller as i grows, it happens often that $M_1 \cap \dots \cap M_i \subset M_{i+1}$. Checking these occurences to avoid the computation of $M_1 \cap \dots \cap M_i$ is both easy and effective.*

5.4 Intersection statistics

There are no standard benchmark for intersections. Meaningful examples may be built by intersect back the ideals produced by the equidimensional decomposition algorithm of [CCT]. For reasons of space, we give not the full examples, for which we refer to [CCT], but we cite our four examples only by name (Gerdt,T2,Butcher,Oaku).
Gerdt is the list of 5 equidimensional ideals, $G_1, \dots, G_5$, produced by the decomposition of the Gerdt ideal. Likewise, T2, Butcher and Oaku are list of 2, 19 and 6 ideals respectively. We compute the intersection of the ideals of the four lists, using repeatedly the intersection algorithm for two ideals.

Headings : **HMod** gives the Hilbert-driven module algorithm timings, **Mod** gives the module algorithm timings, **HRing** gives the Hilbert-driven ring algorithm timings, **Ring** gives the ring algorithm timings and **3** gives the "third" algorithm timings.

Name	HMod	Mod	HRing	Ring	3
Gerdt	4.7 s	12.7 s	5.3 s	10.1 s	13.0 s
T2	0.3 s	0.7 s	0.2 s	0.8 s	0.3 s
Butcher	10.8 s	22.8 s	13.1 s	15.6 s	24.1 s
Oaku	0.5 s	0.5 s	0.7 s	0.5 s	0.4 s

To test our algorithms in a more general setting, we pick an ideal from each of the lists previously considerated and intersect them.

Name	HMod	Mod	HRing	Ring	3
$G_1 \cap T_1$	0.5 s	2.3 s	0.5 s	1.6 s	2.2 s
$G_1 \cap T_1 \cap B_1$	6.2 s	8.3 s	6.4s	7.6 s	8.3 s
$G_1 \cap \cdots \cap O_1$	70 s	164 s	190 s	245 s	1175s

We note that the tag variable module Hilbert-driven algorithm is clearly the best.

Some remarks on the computations :

1. in every case, it is slightly better to start the intersection algorithms with the Gröbner bases as the system of generators of the ideals.
2. The algorithms compute different Gröbner bases, but the Gröbner bases differ very little in the number of polynomials. Usually the ring algorithm basis has one or two less polynomials than the module algorithm.
3. The computation of the Gröbner bases of the ideals has in every instance negligible costs with respect to the costs of the intersection algorithms.
4. The timings are from a prototipical high-level implementation of the algorithms, hence its only interest is for an evaluation of the algorithms.

6 Transporter

The algorithms for the computation of the transporter have apparently more variety than the algorithms for the intersection. These algorithms are not much different for the ideal and the module case, so we describe just the ideal case, i.e. we compute $I : J$, $I := (f_1, \ldots, f_k)$, $J := (g_1, \ldots, g_m)$, where I, J are ideals of $R := k[x_1, \ldots, x_n]$.

We can classify these algorithms in three categories:

1. Algorithms when J is homogeneous and generated by the last variable in DegRevLex ordering:
   (a) Special remark by Bayer.
2. Algorithms when J is principal
   (a) Divided intersection algorithm.
   (b) Tag variable algorithm.
   (c) Syzygy algorithm.
   (d) Reduction to the Bayer special remark (homogeneous ideals only).
3. Algorithms when J has many generators
   (a) Through intersection.
   (b) Through random elements.
   (c) Through a "special generic element".

We want to show that, notwithstanding the number of these algorithms, they are just the same algorithm with small variations.

6.1 Bayer special remark

Assuming that I is homogeneous and J is generated by the last variable x_r in DegRevLex ordering, $I : J$ is obtained dividing by the last variable all the elements of a DegRevLex Gröbner basis of I whose head is divisible by x_r (by the definition of DegRevLex, if f is homogeneous and its head is divisible by x_r then f itself is divisible by x_r).

6.2 Principal quotient

Let $I := (f_1, \ldots, f_k)$ and $J := (g)$.

Divided intersection algorithm
$I : J$ is obtained dividing by g all the elements of $I \cap (g)$.

Tag variable module algorithm
$I : J$ is obtained eliminating the additional (module) variable t from $tI + (tg + 1)$. This algorithm allows us a variant: eliminate t from $tI + I + (tg + 1)$.

Syzygy algorithm
Consider the syzygies of the element $(g, f_1, \ldots, f_k)$ and consider the first component.

Reduction to Bayer special remark
If the ideals I and $J := (g)$ are homogeneous, we add a new variable y of degree $\deg(g)$, and then we compute $(I, g - y) : (y)$ with the Bayer remark algorithm and then eliminate y from the result.

The divided intersection algorithm is equal to the tag variable algorithm or to the syzygy algorithm, depending on the algorithm of your choice for the intersection, with the only difference that one component of all the polynomials is multiplied by g before the algorithm and divided by g at the end. This makes all the computations slower, but the algorithm does not change. We have already seen that the intersection algorithms are the same, hence the three algorithms are substantially the same.

The syzygy algorithm can of course be improved avoiding to compute the components of the syzygies that are discarded at the end; we have hence to compute the Gröbner basis of

$$\begin{pmatrix} f_1 & \ldots & f_n & g \\ 0 & \ldots & 0 & 1 \end{pmatrix}$$

and is hence identical to the tag variable algorithm (if t is taken, as it should, as module variable).

The variant of the tag variables algorithm is convenient if we start from a Gröbner basis: the addition of the generators of I does not require the addition of further critical pairs, and can avoid many computations. For example, when $I : (g) = I$, adding initially a Gröbner basis f_i of I avoids at the end to prove that this same set f_i is a Gröbner basis.

Proposition 8 *Let $R := k[x_1, \ldots, x_n]$, I be an ideal of R and $g \in R$. Then, using the matrix notation for the modules, the Gröbner basis of* $\begin{bmatrix} I & g \\ \mathbf{0} & 1 \end{bmatrix}$ *with respect to the module term-ordering μ is*

$$\begin{bmatrix} \mathrm{GB}((I,g), \mu_{|R}) & \mathbf{0} \\ \mathbf{F} & \mathrm{GB}(I : (g), \mu_{|R}) \end{bmatrix}$$

if μ is a block ordering on $R \oplus R$ for which the first component is the biggest.

The computation of $I : (g)$ with the tag variable module algorithm has as side effect the computation of a Gröbner basis of (I, g); this may be quite useful, see [CCT]. If we happen to know the Hilbert function of (I, g) or $I : (g)$, we have the same possibilities as in the Hilbert-driven intersection algorithm.

6.3 Principal transporter - graded case

If the ideals I and J are homogeneous with respect to a system of weights $\mathbf{w}$, then the transporter algorithms may be easily adapted to keep the homogeneity. It is only necessary to give to the tag variables suitable weights.

For the divided intersection algorithm, follow section 5.2. For the syzygy algorithm, the Hilbert-driven algorithm may be used forthwith. The tag variable algorithm has to be treated with a bit more care.

Proposition 9 *Let $R := k[x_1, \ldots, x_n]$ be graded by the system of weights $\mathbf{w}$, I be a $\mathbf{w}$ homogeneous ideal of R and $g \in R$ $\mathbf{w}$ homogeneous. Let t_1, t_2 be new module variables. Then*

1. *The module $(t_1 I, t_1 g + t_2^{\partial_{\mathbf{w}}(g)+1}) \subset t_1\, R \oplus t_2\, R$ is graded with respect to the system of weights*

$$\mathbf{u} := \left[\begin{array}{c|cc} & 1 & 1 \\ [\boldsymbol{w}] & \vdots & \vdots \\ & 1 & 1 \end{array} \right]$$

2. *If τ is a term-ordering on $R[t_1, t_2]$ fulfilling the conditions of proposition 5, 2. then $(GB(t_1 I, t_1 g + t_2^{\partial_{\mathbf{w}}(g)+1}, \tau))_{|t_2=1} \cap R = I \cap J$.*
3. $\mathbf{H}^{\mathbf{u}}_{(t_1 I, (t_1 g + t_2^{\partial_{\mathbf{w}}(g)+1}))} = \mathbf{H}^{\mathbf{w}}_{(I,g)} + \mathbf{H}^{\mathbf{w}}_{I:(g)} = \mathbf{H}^{\mathbf{w}}_{I}$.

Proposition 9 allows us to use the Hilbert-driven techniques in the transporter algorithm, graded case. As in the intersection algorithm, homogeneization of the input is quite useful in non homogeneous cases.

6.4 Bayer's remark algorithms

The "special remark by Bayer" algorithm just describes what happens when performing the tag variable algorithm in that special situation; if we use the Hilbert-driven Buchberger algorithm; we assume that f_i is already a Gröbner basis. We multiply all the f_i by t, add the equation $tx_r - 1$, then we compute the Gröbner basis; this computation, using the fact that the input is Gröbner, consists in:

1. substitute tx_r with 1 in all the polynomials having tx_r in the head (S-polynomials between tf_i and $tx_r - 1$ when $x_r \mid Lpp(f_i)$).
2. multiply by x_r all the polynomials not having x_r in the head, then substitute tx_r with 1 (S-polynomials between tf_i and $tx_r - 1$ when $x_r \nmid Lpp(f_i)$); this means, put in the result all the polynomials f_i not divisible by x_r unchanged.
3. perform a Buchberger algorithm on the polynomials not having t in the head, just to discover that they form a Gröbner basis.

Just remark that this last step is avoided using the Hilbert function: we already know that no further useful pair exists. The first step is exactly what is done in the "special remark" algorithm, and the second just retrieves f_i first multiplying it by t, then multiplying it by x_r, then dividing it by tx_r. Of course, this algorithm is very convenient if we are in this case.

The "reduction to the special remark" just adds a variable to have a simple computation, then eliminates it; but of course this requires two Gröbner basis computations for these two steps. This is just an indirect way to perform the tag variable algorithm, through an intermediate computation with a different ordering. Whether this is convenient or not is matter of thorough experimentation.

Remark 10 *If g is not homogeneous, and I is homogeneous, we can either homogenize g or split g into its homogeneous components g_i, and divide by the ideal defined by the g_i.*

6.5 Non principal quotient

Now we are left with the case in which J has more than one element. There are three algorithms to consider;

Reduction to principal through intersection: Use the fact that $I : (g_1, \ldots, g_m) = (I : g_1) \cap \cdots \cap (I : g_m)$. A shortcut is possible: if we realize that $((I : g_1) \cap \cdots \cap (I : g_{i-1}))g_i \subseteq I$ we can skip $(I : g_i)$ in the intersection. Note that the check $K \subset (I : g_i)$ may be performed testing if $g_k K \subset I$, see subsection 5.3.

Reduction to principal through random element: Compute $I_b := I : (b)$ where b is a (random) element of J; if $I_b J \subseteq I$ then $I_b = I : J$. The point is that if b is "generic" (whatever this means) then I_b is indeed $I : J$, and the result can be checked: if $I_b J \subseteq I$ then $I_b = I : J$, since in any case $I_b \supseteq I : J$.

Reduction to principal through "special generic" element: Add a new variable y, and consider $b := g_1 +$

$yg_2+\cdots+y^{m-1}g_m$; then $I:(b)$ is generated by elements that do not contain y, and these generate $I:J$. The spirit of the algorithm is that b is sufficiently "generic" to make the previous algorithm work.

The reduction to principal through a random element is not easy algorithm to analyze, just because of the randomness, but it can be considered a variant of the "reduction to principal through intersection" algorithm: b is added as the first generator, and after computing $I:(b)$ we check that the other intersections are not needed (and if not we restart with another random element). The "reduction to principal through a 'special generic' element" algorithm is on one side a form of "reduction to principal through a random element" algorithm in which the element $b := g_1+yg_2+\cdots+y^{m-1}g_m$ is proved to be sufficiently random. On the other side we will see that this algorithm has a lot of relations with the "reduction to principal through intersection" algorithm.

To understand it, first modify the algorithm considering the algorithm not in the ring $k[X, y]$ but in the free $k[\mathbf{x}]$-module $M = k[\mathbf{x}, y]/(y^m)$. Instead of I consider $I \otimes M$ (generated by the $f_i \otimes y^j$), and divide it by the element $b = g_1 + yg_2 + \cdots + y^{m-1}g_m$. This means considering the Gröbner basis of the ideal generated by the columns of the matrix

$$\begin{pmatrix} f_1 & \dots & f_n & 0 & \dots & 0 & \dots & 0 & \dots & 0 & g_m \\ 0 & \dots & 0 & f_1 & \dots & f_n & \dots & 0 & \dots & 0 & g_{m-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & f_1 & \dots & f_n & g_1 \\ 0 & \dots & 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 & 1 \end{pmatrix}$$

with respect to an ordering eliminating all the components except the last one.

Computing instead with the "reduction to principal through intersection" we have first to compute several $I:(g_i)$ e.g. through matrices

$$\begin{pmatrix} f_1 & \dots & f_n & g_m \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

then compute the intersection e.g. through the multivariate form of the "third algorithm". Combining all the computations in one, we have to consider the matrix

$$\begin{pmatrix} f_1 \dots f_n & 0 \dots 0 & \dots & 0 \dots 0 & g_m & 0 & 0 \dots 0 & 0 \\ 0 \dots 0 & 0 \dots 0 & \dots & 0 \dots 0 & 1 & 0 & 0 \dots 0 & 1 \\ 0 \dots 0 & f_1 \dots f_n & \dots & 0 \dots 0 & 0 & g_{m-1} & 0 \dots 0 & 0 \\ 0 \dots 0 & 0 \dots 0 & \dots & 0 \dots 0 & 0 & 1 & 0 \dots 0 & 1 \\ \vdots \ \vdots & \vdots \ \vdots & \ddots & \vdots \ \vdots & \vdots & \vdots & \vdots \ddots \vdots & \vdots \\ 0 \dots 0 & 0 \dots 0 & \dots & f_1 \dots f_n & 0 & 0 & 0 \dots g_1 & 0 \\ 0 \dots 0 & 0 \dots 0 & \dots & 0 \dots 0 & 0 & 0 & 0 \dots 1 & 1 \\ 0 \dots 0 & 0 \dots 0 & \dots & 0 \dots 0 & 0 & 0 & 0 \dots 0 & 1 \end{pmatrix}$$

and it is easy to pass from the one to the other. From the description it is not easy to see which algorithm should be preferred; there are different issues:

- Is it more convenient to use the module or the ring algorithm for the "special generic element" algorithm? While a naive implementation would require a recomputation of the Gröbner basis of I for each component, this of course could be done once and for all. It is however clear that duplications in the intermediate results remain, unless more sophisticated tricks are used. On the other side the module algorithm requires less pairs, and the computation is bounded in degree $< r$ in y (but this can also be obtained adding the generator y^r in the ring case).
- Is it more convenient to collect the results of the different $I:(g_j)$ then intersect them, or to compute the Gröbner basis of the big matrix?
- What is the difference in the practical computation between the smaller and the larger matrix?
- What is the influence of the possibility to use the Hilbert-driven algorithm?

Our guess is that the possibility of using the Hilbert-driven algorithms overwhelms all the possible advantages of the other variants. But this has to be proved in practice.

6.6 Transporter statistics

As in the intersection situation, there are no standard benchmarks for transporter. Here there are some examples. Gredt and Gerdt' come from the example Gerdt of subsection 5.4. C_6 and C_6' are slight modification of the cycylc 6 ideal. We have tested the "reduction to the Bayer remark" Hilbert-driven algorithm and the tag variable Hilbert-driven module algorithm for the principal quotient case.
Bayer gives the timings for the Hilbert-driven reduction to Bayer remark algorithms, while **Htr** gives the timings for the tag-variable Hilbert-driven module algorithm.

Name	**Bayer**	**HTr**
Gerdt	11.7 s	14.9 s
Gerdt'	16.2 s	13.1 s

Name	**Bayer**	**HTr**
C_6	48.2 s	12.5 s
C_6'	23.1 s	12.5 s

In the Gerdt example, the ideal was discovered to be (1) after the DegRevLex computaion in the Bayer remark algorithm.

7 Saturation

There are two main algorithms for the computation of a saturation if the saturating ideal is principal.

Proposition 11 *Let $R := k[x_1, \dots, x_n]$, I be an ideal of R and $g \in R$. Then*

a. there exists $n \in \mathbb{N}$ such that $I:(g)^n = I:(g)^\infty$.

b. If t is a new ring variable, we have that $(tI, tg+1) \cap k[x_1, \dots, x_n] = I:(g)^\infty$.

Using the first part of the proposition, a saturation is computed by several transporter computations. Any transporter algorithm may be used. Using the second part of the proposition, a saturation is computed directly via an elimination Gröbner basis. Which algorithm is the best depends from the input, but usually *a.* is better since it leads to split computations.

If I, g are homogeneous with respect to a system of weights $\mathbf{w}$, it is possible to use the graded variants of the transporter algorithms, or the following

Proposition 12 *Let $R := k[x_1, \ldots, x_n]$ be $\mathbf{w}$-graded, I be a homogeneous ideal of R and g be an homogeneous element of R. Let t_1, t_2 be new ring variables. Then*

a. the ideal $(t_1 I, t_1 g + t_2^{\partial_{\mathbf{w}}(g)+1})$ is homogeneous with respect to the system of weights

$$\mathbf{u} := \left[\begin{array}{c|cc} & 1 & 1 \\ [\boldsymbol{w}] & \vdots & \vdots \\ & 1 & 1 \end{array} \right]$$

b. If τ is a term-ordering fulfilling the conditions of proposition 5, 2. then we have that $(GB(t_1 I, t_1 g + t_2^{\partial_{\mathbf{w}}(g)+1}, \tau))_{|t_2=1} \cap R = I : (g)^\infty$

c. If $d = \partial_{\mathbf{w}}(g) + 1$, then $\langle t_1 I, t_1 g + t_2^d \rangle^{\mathbf{u}} = (1 - \lambda + \lambda \langle I \rangle^{\mathbf{w}})(1 - \lambda^d)$

7.1 Non-principal Saturations

In the general case, there are two main methods to compute a saturation:

1. the repeated transporter method, like in the principal saturation algorithm;
2. the reduction to a principal saturation through some "generic" element.

Since the possible combinations of algorithms for the transporter and saturation (especially in the non-principal case) are many, while almost every algorithm requires a (albeit slight) modification in the code of the Buchberger algorithm, we are still working on these algorithms. Our conjecture is that the repeated intersection/repeated transporter algorithms in their Hilbert-driven versions are the best, since usually it pays to split the computations, but further experiments are needed.

References

[BGS] Bayer, D., Galligo, A., Stillman, M., (1991). *On primary decomposition*, preliminary version of an unpublished paper.

[BS] Bayer, D. and Stillman, M. (1992). *Computation of Hilbert functions*, J. of Symbolic Comp. **14**, 31–50.

[BW] Becker, T. and Weispfenning, W., (1992). *A computational approach to Commutative Algebra.* Graduate Text in Mathematics, *Springer Verlag.*

[BCRT] Bigatti, A.M., Conti, P., Robbiano, L., Traverso, C., (1992). *A "Divide and conquer" algorithm for Hilbert-Poincaré series, multiplicity and dimension of monomial ideals*, Proceedings of AAECC-10 LNCS 673, Springer.

[Bu] Buchberger, B., (1985). *Gröbner bases: An algorithmic method in polynomial ideal theory*, In: Bose, N.K.(ed) *Multidimensional Systems Theory*, Riedel, Dordrecht, pp. 184-232.

[Ca1] Caboara, M., (1997). A modified Buchberger Algorithm for resolutions. Preprint.

[Ca2] Caboara, M., (1998). *Optimization of basic algorithms in Commutative Algebra.* PhD Thesis, Universitdi Pisa.

[CCT] Caboara, M., Conti, P., Traverso, C., (1997). *Yet Another Ideal Decomposition Algorithm*, AAECC-12, Springer LNCS.

[CDR] Caboara, M., De Dominicis G. and Robbiano L., (1996). *Multigraded Hilbert functions and Buchberger algorithm*, Proceeding of the ISSAC96 Conference, Zurich, pp.72-85.

[CS] Caboara, M. and Silvestri M., (1997). *Compatible module orderings.* Submitted.

[CT] Conti P. and Traverso C., (1991). *Computing the conductor of an integral extension*, Discrete applied mathematics 33, no. 1-3, 61–72.

[CLS] Cox D., Little J. and O'Shea D., (1993). *Ideal, Varieties and Algorithms.* Undergraduate Text in Mathematics, *Springer Verlag.*

[Eis] Eisenbud D., (1994). *Commutative Algebra with a View Toward Algebraic Geometry*, Springer Verlag, Graduate Text in Mathematics.

[GM] Gebauer, R. and Möller M., (1988). *On an installation of the Buchberger algorithm*, J. Symbolic Computation, **6**/2/3 pp. 275-286.

[GTZ] Gianni, P., Trager B. and Zacharias G., (1988). *Gröbner bases and primary decomposition of polynomial ideals*, J. Symbolic Computation, **6**/2/3 pp. 149-167.

[MM] Möller, H.M. and Mora F., (1986). *New constructive methods in classical ideal theory.* J. Algebra **100**, pp. 138-178.

[RR] Rust C.J. and Reid G.J., (1997). *Rankings of partials derivatives*, Proceedings of ISSAC'97, pp.9-16, 1997.

[Sch] Schreyer F.O., (1980). *Die Berchnung von Syzygien mit dem verallgemeinerten Weierstrass'schen Divisionsatz*, Diplomarbeit, Hamburg.

[Sp] Spear, D.A., (1977).*A constructive approach to commutative ring theory*, In: *Proc, MACSYMA User's conference*, NASA CP-2012, pp. 369-376.

[TD] Traverso, C. and Donati L., (1989). *Experimenting the Gröbner basis algorithm with the ALPI system*, In:ISSAC '89, ACM press, NewYork, pp. 192-198.

[Wa] Wall, B., (1989). *On the computation of syzygies*, ACM SIGSAM Bullettin **23**/4 pp. 5-14.

[Tr] Traverso C., (1996). *Hilbert functions and the Buchberger algorithm*, Journal of Symbolic Computation, 22-4 pp.355-376, 1996.