IMPLEMENTING THE ALGORITHM FOR COMPUTING FORMAL MODULI

ARVID SIQVELAND

0. INTRODUCTION

Given a commutative k-algebra A and an A-module M. In [Siq 1] we give the appropriate definitions of the local formal moduli, i.e. the prorepresenting hull $\hat{H}_M$ of the deformation functor Def_M. In [La 1] , Laudal gives the theory of computing $\hat{H}_M$, and the algorithm 0.1 can be found in [Siq 1].

algorithm 0.1.

Begin

!All variables start with empty;

Put $\bar{B} = \{\underline{n} \in N^d \mid |\underline{n}| \leq 1\}$*;*

put $B = \{\underline{n} \in N^d \mid |\underline{n}| = 1\}$*;*

Put $\alpha_{e_i} = x_i^\star$ *and* $\alpha_{\underline{0}} = d$*;*

While not finished do

begin $i := i + 1$*;*

Pick a basis B'_{i+1} *for* $\underline{m}^{i+1}/\underline{m}^{i+2} + \underline{m}^{i+1} \bigcap \underline{m}(f_1, ..., f_r)$ *such that for each* $\underline{n} \in B'_{i+1}$ *we have* $u^{\underline{n}} = u_k \cdot \underline{u}^{\underline{m}}$ *for some* $\underline{m} \in B$*;*

Put $\bar{B}'_{i+1} = \bar{B} \bigcup B'_{i+1}$*; Then for each* $\underline{n} \in N^d$ *with* $|\underline{n}| \leq i+1$ *we have a unique relation in* $k[\underline{u}]/\underline{m}^{i+2} + \underline{m}(f_1, ..., f_r)$*, namely*

$\underline{u}^{\underline{n}} = \sum_{\underline{m} \in \bar{B}'_{i+1}} \beta'_{\underline{m},\underline{n}} \underline{u}^{\underline{m}} + \sum_j \beta_{\underline{n},j} f_j$*. For each* $\underline{n} \in B'_{i+1}$ *compute*

$y(\underline{n}) = \sum_{|\underline{m}| \leq i+1} \sum_{\underline{m}_1 \underline{m}_i \in \bar{B}} {}_{+\underline{m}_2 = \underline{m}} \beta'_{\underline{m},\underline{n}} \alpha_{\underline{m}_1} \circ \alpha_{\underline{m}_2}$

Put $f_j = f_j + \sum_{\underline{n} \in B'_{i+1}} y_j(\overline{y(\underline{n})}) \underline{u}^{\underline{n}}$

Pick a basis B *for* $\underline{m}^{i+1} + (f_1^i, ..., f_r^i)/\underline{m}^{i+2} + (f_1, ..., f_r)$ *such that* $B \subseteq B'_{i+1}$ *($f_\cdot^i$ = sum up to degree i)*

Put $\bar{B} := \bar{B} \bigcup B$*; then, for each* $\underline{n} \in N^d$ *with* $|\underline{n}| \leq i+1$ *we have a unique relation in* $k[u]/\underline{m}^{i+2} + (f_1, ..., f_r)$*, namely*

$\underline{u}^{\underline{n}} = \sum_{\underline{m} \in \bar{B}} \beta_{\underline{n},\underline{m}} \underline{u}^{\underline{m}}$.

Pick for each $\underline{m} \in B$ *an* $\alpha_{\underline{m}} \in Hom^1_B(L., L.)$ *such that*

$d\alpha_{\underline{m}} = -\sum_{l=0}^{i+1-2} \sum_{\underline{n} \in B'_{2+l}} \beta_{\underline{n},\underline{m}}\, y(\underline{n})$*;*

End;

End;

In this paper our goal is to make a program in Singular [Si] implementing this algorithm. We will solve one problem at time refering to procedures building up a Singular library given in paragraph 3, and give the complete program in the same

paragraph. Thus I will not write up the procedures where they naturally belongs but just refer to paragraph 3.

I would like a comment on my programming philosophy. It is known that Singular doesn't like a lot of procedures in the scripts. It is also known that if you don't build up the program with objects and procedures, debugging will be impossible. As long as it runs sufficiently fast, I have decided to use procedures. Thus the program (proc moduli) should be possible to read. When extra speed is needed, it will be possible to put the procedure scripts into the program. The resulting library will be available via e-mail:Arvid.Siqveland@hibu.no

1. General Philosophy

When implemented directly, algorithm 0.1 is too ineffective to give results of interest. Thus we will give some comments on how to make it more effective. The idea is to not compute things that we know will be identically zero and to record only the complements when choosing monomial bases. We will assume that we have $\dim_k(\mathrm{Ext}^1_A(M,M)) = d$ with basis $\{x_i^\star\}_{i=1}^d$ so that we are working in the formal power series ring $k[[u_1,...,u_d]]$ with maximal ideal $\underline{m} = (u_1,..,u_d)$.

1.1. Choosing monomial bases. In algorithm 0.1 there are two places where monomial bases are choosen:

i) Pick a basis B'_{i+1} for $\underline{m}^{i+1}/\underline{m}^{i+2} + \underline{m}^{i+1} \bigcap \underline{m}(f_1,...,f_r)$ such that for each $\underline{n} \in B'_{i+1}$ we have $u^{\underline{n}} = u_k \cdot \underline{u}^{\underline{m}}$ for some $\underline{m} \in B$

ii) Pick a basis B for $\underline{m}^{i+1} + (f_1^i,...,f_r^i)/\underline{m}^{i+2} + (f_1,...,f_r)$ such that $B \subseteq B'_{i+1}$ ($f_\cdot^i$ = sum up to degree i)

Noticing that

$$\underline{m}^{i+1} + (f_1^i,...,f_r^i)/\underline{m}^{i+2} + (f_1,...,f_r) \cong \underline{m}^{i+1}/\underline{m}^{i+2} + (f_1,...,f_r),$$

we can say that the problem in both cases is:

Consider the k-algebra $A = k[[u_1,...,u_d]]$ with maximal ideal $\underline{m}$. Let $\alpha \subseteq \underline{m}^n$, $n \geq 1$ and $\beta \subseteq \underline{m}^2$ be ideals. Assume there is given a linear span of the k-vectorspace

$$\alpha/(\underline{m}^{n+1} + \beta) \cap \alpha = V_\beta$$

on the form $\{\underline{u}^{\underline{n}}\}_{\underline{n} \in S}$. Choose a monomial basis for V_β on the form

$$\{\underline{u}^{\underline{n}}\}_{\underline{n} \in B} \text{ with } B \subseteq S.$$

The naive way to do this is the following: If $S = \{\underline{u}^{\underline{m}_1},...,\underline{u}^{\underline{m}_r}\}$, then we fill out B one by one element by the knowledge that $\underline{u}^{\underline{m}_{p+1}}$ is linearly dependent of $\{\underline{u}^{\underline{m}_1},...,\underline{u}^{\underline{m}_p}\}$ if and only if $\underline{u}^{\underline{m}_{p+1}} = 0$ in

$$A/m^{n+1} + \beta + (\underline{u}^{\underline{m}_1},...,\underline{u}^{\underline{m}_p}).$$

This is ineffective because the "one-by-one"-statement involves a lot of standard basis computation. It turns out that computing complementary bases is much more effective.

To save memory (and it turns out to speed up), we let

$$S^C = \{\underline{n} \in \mathbf{N} : |\underline{n}| = i+1\} - S,$$

and we choose a set of tuples B^C such that $\{\underline{u}^{\underline{n}}\}_{\underline{n}\in B^C}$ is the monomials not in a basis and such that $B^C \supseteq S^C \Rightarrow B \subseteq S$.

Notice now that this way of recording monomial bases gives us an extra bonus when computing Massey products. The only monomials that are related to others are exactly those in B^C. The algorithm for computing complementary bases, is trated in section 2.2.

1.2. Computing relations.

In algorithm 0.1 there are two places where we need the relations:

i) for each $\underline{n} \in \mathbf{N}^d$ with $|\underline{n}| \leq i+1$ we have a unique relation in $k[\underline{u}]/\underline{m}^{i+2} + \underline{m}(f_1, ..., f_r)$, namely

$$\underline{u}^{\underline{n}} = \sum_{\underline{m}\in \bar{B}'_{i+1}} \beta'_{\underline{m},\underline{n}} \underline{u}^{\underline{m}} + \sum_j \beta_{\underline{n},j} f_j.$$

ii) for each $\underline{n} \in \mathbf{N}^d$ with $|\underline{n}| \leq i+1$ we have a unique relation in $k[u]/\underline{m}^{i+2} + (f_1, ..., f_r)$, namely

$$\underline{u}^{\underline{n}} = \sum_{\underline{m}\in \bar{B}} \beta_{\underline{n},\underline{m}} \underline{u}^{\underline{m}}.$$

In both cases, these relations can be recorded in a matrix R with the following property: Assume that B^C is a complementary basis for $k[\underline{u}]/Q$ as above. Let $B(i)$ be the i'th element (monomial) in B, respectively, let $B^C(j)$ be the j'th element in B^C. Then $B(i)^\star(B^C(j)) = R(i, j+1)$. The first column in R is filled out with 1's for computational reasons to illustrate that $B(i)^\star(B(i)) = 1$. This is done in section 2.4.

1.3. Some Comments on the Use of 1.1 and 1.2.

Looking at the sentence

"For each $\underline{n} \in B'_{i+1}$ compute
$y(\underline{n}) = \sum_{|\underline{m}|\leq i+1} \sum_{\underline{m}_1 \atop \underline{m}_i \in \bar{B}} {}_{+\underline{m}_2=\underline{m}} \beta'_{\underline{m},\underline{n}} \alpha_{\underline{m}_1} \circ \alpha_{\underline{m}_2}$"

in algorithm 0.1, we see that it is enough to join in the sum the elements where there exists a relation and where the mentioned $\alpha_{\underline{m}}$'s are different from zero. Thus

it is only necessary to store the α's different from zero, and the computation is as fast as it can be.

Looking at the sentence

"Pick for each $\underline{m} \in B$ an $\alpha_{\underline{m}} \in Hom^1_B(L.,L.)$ such that
$d\alpha_{\underline{m}} = -\sum_{l=0}^{i+1-2} \sum_{\underline{n} \in B'_{2+l}} \beta_{\underline{n},\underline{m}}\, y(\underline{n})$;"

in algorithm 0.1, more care must be taken. We have not recorded any $y(\underline{n})$ that is different from zero, and the choosen $\alpha_{\underline{m}}$ will automatically be set to zero. Also, if the sum is identically zero, $\alpha_{\underline{m}}$ will be set to zero. But this is just the first element in the family $\{\alpha_{\underline{m},p}\}_{p\geq 2} \in \mathrm{Hom}^2_A(L.,L.)$. Can we be shure that $\alpha_{\underline{m},2}$ can be the first element in a family fulfilling the condition $d\alpha_{\underline{m}} = -\sum_{l=0}^{i+1-2} \sum_{\underline{n} \in B'_{2+l}} \beta_{\underline{n},\underline{m}}\, y(\underline{n})$. The answer to this is yes, and the reason is lemma 2.1.

1.4. Computing Massey products in algorithm 0.1.

Going back to algorithm 0.1, we understand that we need a way to treat the differential graded k-algebra $Hom(L.,L.)$. This is solved by the following:

Proposition.

In the leading algorithm, it is enough to let every element $\alpha \in \mathrm{Hom}^1(L.,L.)$ be represented by the couple $\{\alpha_1, \alpha_2\}$ where $\alpha = \{\alpha_i\}_{i>0}$.

Proof.

Because the isomorphism $H^2(Hom(L.,L.)) \longrightarrow \mathrm{Ext}^2_A(M,M)$ sends $\overline{\alpha} = \overline{\{\alpha_i\}}_{i>0}$ to $\overline{\alpha_1 \circ \alpha_0}$ with

$$\alpha_1 \circ \alpha_0 \in \mathrm{Hom}(L_2, L_0) \longrightarrow \mathrm{Hom}(L_2, M)$$

it is obvious that this representation is enough to compute Massey products when a defining system is given. This will give the wanted element in $\mathrm{Ext}^2_A(M,M) \cong H^2(Hom(L.,L.))$. The main problem is exactly choosing defining systems. To be precise, this is the following problem: Given a coboundary

$$\xi = \{\xi_i\}_{i\geq 2} \in \mathrm{Hom}^2(L.,L.).$$

Choose an $\alpha \in \mathrm{Hom}^1(L.,L.)$ such that $d(\alpha) = \xi$. Now we only know ξ_2, and in fact, we are only interested in α_1 and α_2. Now it turns out that finding α_1 and α_2 is possible, see section 2.3, and lemma 2.1 in the same section states that this is the two first morphisms in an α such that $d(\alpha) = \xi$.

2. The Different Parts

2.1 Ext-Computation.

The first problem is to compute a basis for $\mathrm{Ext}^i_A(M,M)$, $i = 1,2$. In fact we want a basis for

$$\mathrm{Ext}^i_A(M,M) \cong H^i(Hom(L.,L.)),\ i = 1,2,$$

where $L.$ is a fixed free resolution of M and $Hom(L.,L.)$ is the complex defined by

$$Hom^p_A(L.,L.) = \prod_{m\geq p} Hom_A(L_m, L_{m-p}),$$

$d^p : Hom_A^p(L.,L.) \longrightarrow Hom_A^{p+1}(L.,L.)$ given by

$$d^p(\{\alpha_i^p\}) = \{d_i \circ \alpha_{i-1}^p - (-1)^p \alpha_i^p \circ d_{i-p}\}.$$

Notice that this gives $Hom(L.,L.)$ the essential property of beeing a differential graded k-algebra [Siq 1]. We start by finding a basis for $H^i(\mathrm{Hom}(L.,M))$: Given a free resolution

$$0 \longleftarrow M \longleftarrow A^{n_0} \xleftarrow{d_1} A^{n_1} \xleftarrow{d_2} A^{n_2} \xleftarrow{d_3} \ldots.$$

This implies the following commutative diagram

$$\begin{array}{ccccccccc}
\longrightarrow & \mathrm{Hom}(A^{n_0},M) & \longrightarrow & \mathrm{Hom}(A^{n_1},M) & \longrightarrow & \mathrm{Hom}(A^{n_2},M) & \longrightarrow & \cdots \\
 & \cong\uparrow & & \cong\uparrow & & \cong\uparrow & & \\
\longrightarrow & M^{n_0} & \xrightarrow{d_1^T} & M^{n_1} & \xrightarrow{d_2^T} & M^{n_2} & \xrightarrow{d_3^T} & \cdots
\end{array}$$

As is known, $\mathrm{Ext}_A^i(M,M) = \ker d_{i+1}^T / \operatorname{Im} d_i^T$.

Now, in Singular every module is given by its first differential in a resolution. Thus we need a way of exponentiating modules.

Let

$$P \subseteq M,\ Q \subseteq N$$

be sub A-modules. Then we have the exact sequence

$$0 \longrightarrow P \oplus Q \longrightarrow M \oplus N \xrightarrow{\phi} M/P \oplus N/Q \longrightarrow 0.$$

For Singular applications this yields

$$(A^n/ < m_1, m_2, ..., m_r >)^s \cong$$
$$A^{sn}/ < (m_1,0,...,0),(m_2,0,...,0),...,((m_r,0,...,0),...,(0,...,0,m_1),...,(0,...,0,m_r) > .$$

Thus the algorithm for exponentiating a module, is given by the procedure "freerep" in paragraph 3.

To compute a basis for $\mathrm{Ext}_A^i(M,M)$, our next problem is to compute the kernel of a morphism

$$A^n/I_1 \xrightarrow{\phi} A^m/I_2.$$

This is done by computing the kernel in

$$A^n \xrightarrow{\phi} A^m/I_2$$

by the singular command modulo(ϕ , I_2) and then resolve the result modulo I_1. This is given by the procedure "Ker" in paragraph 3. In fact we just need generators for the kernel and so it is unnecessary to reduce modulo I_1. What we need is given in the procedure "augmentker".

The thing left for computing Ext's is to divide by the image of the "previous morphism". The general situation is the diagram

$$\begin{array}{ccccc} R^{n_1}/P_1 & \xrightarrow{A} & R^{n_2}/P_2 & \xrightarrow{B} & R^{n_3}/P_3 \\ \uparrow & & \uparrow & \nearrow_{B'} & \\ R^{n_1} & \xrightarrow[A']{} & R^{n_2} & & \end{array}$$

Thus our result is

$$(\ker B'/P_2)/(\operatorname{Im} A'/P_2) \cong \ker B'/(P_2 + \operatorname{Im} A').$$

This gives rise to the procedure "quotkerim".

Consider $M = A^p/P$. In the diagram

$$\begin{array}{ccc} \operatorname{Hom}(R^m, M) & \xrightarrow{A} & \operatorname{Hom}(R^n, M) \\ \cong\uparrow & & \uparrow\cong \\ M^m & \xrightarrow{A^T} & M^n \\ \cong\uparrow & & \uparrow\cong \\ A^{pm}/\sum_{i=1}^m P & \xrightarrow{F(A^T)} & A^{pn}/\sum_{i=1}^n P \end{array},$$

what does the matrix $F(A^T)$ look like? Well that is pretty obvious, and the result is given in procedure "freerep" in paragraph 3.

Now Ext can be computed from the given resolution by quotkerim. This is done in the procedure "Ext". Anyway this just give Ext as a quotient of a free module, so to get generators of Ext in $\ker d_{i+1}^T/\operatorname{Im} d_i^T$ we map it onto this by procedure "ontoExt".

To find a k-basis for $\ker d_{i+1}^T/\operatorname{Im} d_i^T$, we use the Singular command kbase on "Ext" and map this basis to $\ker d_{i+1}^T/\operatorname{Im} d_i^T$ by "ontoExt". This operation is called procedure "basisExt".

Now our last problem in the Ext computation is to find the corresponding basis elements in the differential graded k-algebra $Hom(L., L.)$. We know how the quasi homomorphism

$$\operatorname{Hom}^p(L., L.) \longrightarrow \operatorname{Hom}(L., M)$$

is given, so we look at the diagram

$$\begin{array}{ccccccccc} & & L_p & \xleftarrow{d_{p+1}} & L_{p+1} & \xleftarrow{d_{p+2}} & L_{p+2} & \xleftarrow{d_{p+3}} & \cdots \\ & & \downarrow{\scriptstyle \alpha_p} & & \downarrow{\scriptstyle \alpha_{p+1}} & & \downarrow{\scriptstyle \alpha_{p+2}} & & \\ M & \xleftarrow[\varepsilon]{} & L_0 & \xleftarrow[d_1]{} & L_1 & \xleftarrow[d_2]{} & L_2 & \xleftarrow[d_3]{} & \cdots \end{array}$$

Then we understand that it is enough to lift the composition $\alpha_{p+l-1} \circ d_{p+l}$ at each step with the right sign (refering to the sign in the definition of the differential in $Hom(L., L.)$). This is easily done by the Singular Command "lift", and the result is given in procedure "alfa".

This completes our Ext-computation.

2.2. Computing monomial bases.

Recall that the problem is the following: Given the k-algebra $A = k[[u_1, ..., u_d]]$ with maximal ideal $\underline{m}$. Let $\alpha \subseteq \underline{m}^i$, $i \geq 1$ and $q \subseteq \underline{m}^2$ be ideals. Assume there is given a linear span of the k-vectorspace

$$\alpha/(\underline{m}^{i+1} + q) \cap \alpha = V_q$$

on the form $\{\underline{u}^{\underline{n}}\}_{\underline{n} \in S}$. Choose a monomial basis for V_q on the form

$$\{\underline{u}^{\underline{n}}\}_{\underline{n} \in B} \text{ with } B \subseteq S.$$

We will use as input the integer i, the ideal q and the complementary span $S^C = \{\underline{n} \in \mathbf{N}^d : |\underline{n}| = i\} - S$ and we will give as output the complementary bases $B^C = \{\underline{n} \in \mathbf{N}^d : |\underline{n}| = i\} - B$. The idea is simply the following:

Having S^C we also have S. Write up the (unique) normal form of S modulo a standardbasis for $\underline{m}^{i+1} + q$. Then some elements may be removed form S, and we add them to S^C. The result is B^C, and the procedure is called "proc basisfrom".

2.3. Lifting of cosycles.

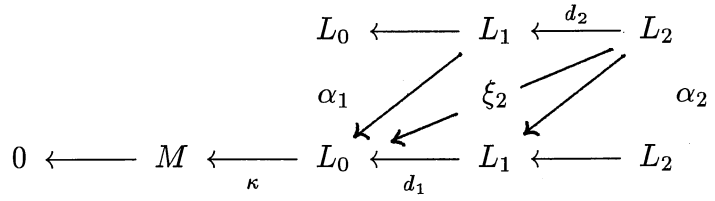
Let $\xi = \{\xi_i\}_{i \geq 2} \in \mathrm{Hom}^2(L., L.)$ be a coboundary. We want an algorithmic (and in Singular implementable) way to find an element

$$\alpha = \{\alpha_i\}_{i \geq 1} \in \mathrm{Hom}^1(L., L.)$$

such that $d(\alpha) = \xi$.

In the computation, given ξ, we only need the first two α_1 and α_2. But we have to prove that this in fact is the first two in a lifting. This leaves us with the problem of finding the first two "that can be lifted". This will be clearer in a moment.

Consider the diagram



We want to find α_1 and α_2 such that

$$\alpha_1 d_2 + d_1 \alpha_2 = \xi_2 \Longleftrightarrow \alpha_1 d_2 - \xi_2 = -d_1 \alpha_2.$$

Thus our first goal is to find α_1 such that $\mathrm{Im}(\alpha_1 d_2 - \xi_2) \subseteq \mathrm{Im}(d_1)$. This can be done much in the same way as in the computation of Ext.

$$\mathrm{Im}(\alpha_1 d_2 - \xi_2) \subseteq \mathrm{Im}(d_1) \Longleftrightarrow \kappa(\alpha_1 d_2 - \xi_2) = 0 \Longleftrightarrow \kappa(\alpha_1 d_2) = \kappa(\xi_2).$$

We have the following diagram

$$
\begin{array}{ccccc}
\mathrm{Hom}(L_0,M) & \xrightarrow{d_1^T} & \mathrm{Hom}(L_1,M) & \xrightarrow{d_2^T} & \mathrm{Hom}(L_2,M) \\
\cong\uparrow & & \cong\uparrow & & \cong\uparrow \\
M^{n_0} & \xrightarrow{d_1^T} & M^{n_1} & \xrightarrow{d_2^T} & M^{n_2} \\
 & & \omega & & \omega \\
 & & \alpha_1 & & \xi_2
\end{array}
\tag{2.1}
$$

Thus finding α_1 can be solved by a lift procedure. Assume that we have found α_1 such that $\mathrm{Im}(\alpha_1 d_2 - \xi_2) \subseteq \mathrm{Im}(d_1)$. Then we can find α_2 such that

$$d_1\alpha_2 = -(\alpha_1 d_2 - \xi_2) \Longrightarrow \alpha_1 d_2 + d_1\alpha_2 = \xi_2.$$

Our theoretical problem is solved by the following

Lemma 2.1.
Suppose found α_{i-1}, α_i such that

$$\alpha_{i-1}d_i + d_{i-1}\alpha_i = \xi_i.$$

Then there exist an α_{i+1} fulfilling the condition, i.e. $d_i\alpha_{i+1} + \alpha_i d_{i+1} = \xi_{i+1}$.

Proof.
Suppose found α_{i-1}, α_i such that $\alpha_{i-1}d_i + d_{i-1}\alpha_i = \xi_i$. We then see that

$$
\begin{aligned}
d_{i-1}(\alpha_i d_{i+1} - \xi_{i+1}) &= d_{i-1}(\alpha_i d_{i+1}) - d_{i-1}\xi_{i+1} \\
&= d_{i-1}(\alpha_i d_{i+1}) - \xi_i d_{i+1} = (-\alpha_{i-1}d_i + \xi_i)d_{i+1} - \xi_i d_{i+1} \\
&= \xi_i d_{i+1} - \xi_i d_{i+1} = 0.
\end{aligned}
$$

Thus there exist an α_{i+1} such that

$$d_i\alpha_{i+1} = \xi_{i+1} - \alpha_i d_{i+1} \Longleftrightarrow d_i\alpha_{i+1} + \alpha_i d_{i+1} = \xi_{i+1}.$$

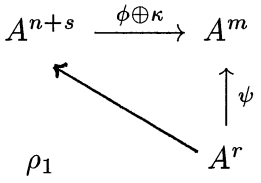
Thus by induction, the whole system may be lifted.

We understand that to make a Singular program, we need a lifting algorithm. Singular only contains the lifting procedure for free modules. We will also need a lifting procedure for quotient modules, as seen in the diagram (2.1). That is, given the diagram

$$
\begin{array}{ccc}
A^n & \xrightarrow{\bar{\phi}} & A^m/P \\
 & \nwarrow & \uparrow\bar{\psi} \\
\rho & & A^r
\end{array}
$$

Such that $\operatorname{Im}\bar{\psi} \subseteq \operatorname{Im}\bar{\phi}$. Then there exist a morphism ρ as shown. This ρ can be found as follows:

Let P be given by $A^s \xrightarrow{\kappa} P \longrightarrow 0$. Considering

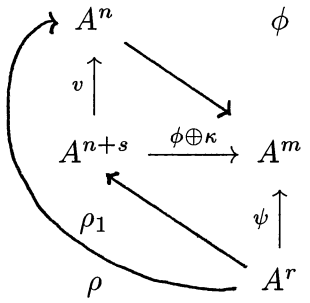


we have

$$\operatorname{Im}\bar{\psi} \subseteq \operatorname{Im}\bar{\phi} \Longrightarrow \operatorname{Im}\psi \subseteq \operatorname{Im}\phi + P = \operatorname{Im}\phi + \operatorname{Im}\kappa = \operatorname{Im}(\phi + \kappa).$$

Thus a ρ_1 as indicated exists and can be found by the Singular command "lift".

Finally, from the diagram



we find

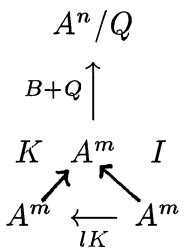
$$\phi(\rho(x)) = \phi(v(\rho_1(x))) = (\phi \oplus \kappa)(\rho_1(x)) - \kappa(x) = \psi(x) - \kappa(x) \Longrightarrow \bar{\phi} \circ \rho = \bar{\psi}.$$

Thus the problem is solved by finding ρ_1 and ϕ by "lift".

This is the basis for procedure "qlift" in paragraph 3, leading to the procedure falfa for lifting cosycles.

2.4. Computing duals.

To compute duals, we use the procedure lift_kbase made by Bernd Martin in the library "Deform.lib". First of all we consider a basis as the columns in a matrix B. The quotient of a free module is represented by the matrix Q whose columns are the generators of the module. We let K denote the basis that Singular chooses, and lK the lifted basis. Then the problem is solved from the following diagram.



The inverse matrix of lK gives all the relations, it is the transformation matrix. We compute this in "proc relyp".

1.5.Knowing when to Stop.

First of all, to know when to stop is no problem :
A defining system $\{\alpha_{\underline{m}}\}_{\underline{m}\in\bar{B}_i}$ corresponds to morphisms

$$d_i^S : L_i \otimes_k S \longrightarrow L_{i-1} \otimes_k S,$$

where $S = k[[\underline{u}]]/\underline{m}^{i+1} + (f_1^i, ..., f_r^i)$. This induces morphisms

$$d_i' : L_i \otimes_k k[[\underline{u}]]/(\underline{f}^i) \longrightarrow L_{i-1} \otimes_k k[[\underline{u}]]/(\underline{f}^i).$$

If $d_i' \circ d_{i-1}' = 0$, we know that d_i' can be lifted to the top and and $d_i = \varinjlim d_i'$ is the proversal family. Going into algorithm 0.1, we see that if $d_1' \circ d_0' = 0$, every obstruction is zero and the choosen α mapping to the obstruction may be choosen to be zero by lemma 2.1. Thus a sufficient condition for stopping the process is $d_1' \circ d_0' = 0$ Also, we can solve the stopping problem in a more combinatorial way.

Proposition 2.2. *In the computation of formal moduli, with the notation from* [Siq 1], *assume that* $(\underline{f}) \subseteq \underline{m}^2 - \underline{m}^{q+1}$ *and that*

$$\alpha_{\underline{m}} = 0 \text{ for } |\underline{m}| \geq \frac{1}{2}(5 + i - 2q).$$

Then the next α*'s are* 0 *and there are no changes in* $(\underline{f})$.

Proof.

$$y(\underline{n}) = \sum_{|\underline{m}|\leq i+1} \sum_{\substack{\underline{m}_1+\underline{m}_2=\underline{m} \\ \underline{m}_i\in\bar{B}_i}} \beta'_{\underline{m},\underline{n}} \alpha_{\underline{m}_1} \circ \alpha_{\underline{m}_2},$$

where $\beta'_{\underline{m},\underline{n}}$ are the relations in $k[[\underline{u}]]/\underline{m}^{i+2} + \underline{m}(\underline{f})$. Now $(\underline{f}) \subseteq \underline{m}^2 - \underline{m}^q$ implies that $\beta'_{\underline{m},\underline{n}} = 0$ for $|\underline{m}| \leq 2 + (i + 1 - q) = 3 + i - q$. Thus for the monomials in the sum above

$$|\underline{m}_1| + |\underline{m}_2| = |\underline{m}| \geq 3 + i - q \Longrightarrow |\underline{m}_i| \geq \frac{1}{2}(3 + i - q).$$

So $y(\underline{n}) = 0$ for $|\underline{n}| = i + 1$ if $\alpha_{\underline{m}} = 0$ for $|\underline{m}| \geq \frac{1}{2}(3 + i - q)$. In particular

$$\frac{1}{2}(3+i-q) \geq \frac{1}{2}(5+i-2q) \Longrightarrow y(\underline{n}) = 0 \text{ for } |\underline{n}| = i+1 \Longrightarrow (\underline{f}) \text{ stays in } \underline{m}^2 - \underline{m}^{q+1}.$$

The next order $\alpha_{\underline{m}}$'s are given by choosing elements $\alpha_{\underline{m}} \in \mathrm{Hom}^1(L., L.)$ mapping to

$$\beta_{\underline{m}} = \sum_{l=0}^{i+1-2} \sum_{\underline{n}\in B'_{2+l}} \beta_{\underline{n},\underline{m}} y(\underline{n}), \ \underline{m} \in B_{i+1}$$

where $\beta_{\underline{n},\underline{m}}$ are the relations in $k[[\underline{u}]]/\underline{m}^{i+2} + (f_1, ..., f_r)$. Then

$$\beta_{\underline{n},\underline{m}} = 0 \text{ for } |\underline{n}| \leq 3 + i - q$$

and

$$|\underline{n}| \geq 3+i-q \Longrightarrow |\underline{n}| \geq 1+(2+i-q) \Longrightarrow y(\underline{n}) = 0,$$

because $\alpha_{\underline{m}} = 0$ for $|\underline{m}| \geq \frac{1}{2}(3+(2+i-q)-q) = \frac{1}{2}(5+i-2q)$. Thus the next order α's can be choosen identically 0.

None of the stopping principles are filled into each loop. To reduce the use of time, the procedure runs a determined number of loops, and then finally tests if this number is enough. The test is $d_1' \circ d_0' = 0$. The moduli computation ends in the procedure "moduli(module M,integer number of loops,eventually the matrices for $d1$, d_2".

3. The Program

```
proc sumideal(list #)
parameters (ideal i1,ideal i2)
{
    ideal @i;
    for(int @j=1;@j<=ncols(#[1]);@j=@j+1)
    {@i[@j]=#[1][@j];};
    for(@j=1;@j<=ncols(#[2]);@j=@j+1)
    {@i[ncols(#[1])+@j]=#[2][@j];};
    @i= removezeros(@i);
    return(@i);
};



proc removezeros(ideal t2)
{
  ideal t1;int n,it,u;n=ncols(t2);
  for(u=1;u<=n;u=u+1)
  {
         if(t2[u]!=0)
     {
          it=it+1;
          t1[it]=t2[u];
         };
   }
   return(t1);
};

proc remove2(list #)
Parameters (i1,i2). Removes the elements in i1 from i2.
{
  int @k1,@k2;ideal @i3,@i4,@i5;
  @i3=#[2];ideal @i1=#[1];
  for(int @j1=1;@j1<=ncols(@i1);@j1=@j1+1)
  {
    @k1=findi(#[1][@j1],@i3);
    if(@k1 !=0)
    {
      if(@k1==1 or @k1==ncols(@i3))
      {
        if(@k1==1)
        {
          if(ncols(@i3)==1)
          {
            @i3=0;
          }
          else
          {
            @i3=@i3[2..ncols(@i3)];
          };
        }
        else
        {
```

```
          @i3=@i3[1..(ncols(@i3)-1)];
        };
      }
      else
      {
        @i4=@i3[1..@k1-1];
        @i5=@i3[@k1+1..ncols(@i3)];
        @i3=sumideal(@i4,@i5);
      };
    };
  };
  return(@i3);
};

proc qlift(list #)
Parameters (module M, matrix A, matrix B).
The modules are given as their first differentials, the procedure
else is exactly the same as the ordinary lift, exept
that it also lifts quotient-modules.
{
  matrix @m=matrixofmodule(#[1]);
  matrix @apm[nrows(#[2])][ncols(#[2])+ncols(@m)];
  @apm[1..nrows(#[2]),1..ncols(#[2])]=#[2];
  @apm[1..nrows(#[2]),ncols(#[2])+1..ncols(#[2])+ncols(@m)]=@m;
  matrix @p=lift(@apm,#[3]);
  matrix @proj[ncols(#[2])][ncols(#[2])+ncols(@m)];
  for(int @i=1;@i<=ncols(#[2]);@i=@i+1)
  {
    @proj[@i,@i]=1;
  };
  return(@proj*@p);
};


proc falfa1(list #)
See help text for falfa.
{
  matrix @c1=vectorofmatrix(#[1]);
  matrix @c2=freerep(#[2],transpose(#[3]));
  matrix @c3=qlift(moduleexp(#[2],ncols(#[3])),@c2,@c1);
  matrix @c4=matrixofvector(@c3,nrows(#[2]),ncols(#[2]));
  return(@c4);
};

proc falfa2(list #)
See help text for falfa.
{
  matrix @c1=vectorofmatrix(#[1]);
  matrix @c2=freerep(#[2],transpose(#[3]));
  matrix @c3=qlift(moduleexp(#[2],ncols(#[3])),@c2,@c1);
  matrix @c4=matrixofvector(@c3,nrows(#[2]),ncols(#[2]));
  matrix @c5=-1*(@c4*#[3]-#[1]);
  matrix @c6=lift(#[2],@c5);
  return(@c6);
};


proc vectorofmatrix
```

```
{
  matrix @c[ncols(#[1])*nrows(#[1])][1];
  int @i;int @j;
  for (@j=1;@j<=ncols(#[1]);@j=@j+1)
  {
    for (@i=1;@i<=nrows(#[1]);@i=@i+1)
    {
      @c[@i+(nrows(#[1]))*(@j-1),1]=#[1][@i,@j];
    };
  };
  return(@c);
};

proc svectorofmatrix
{
  vector @c;
  int @i;
  for (@i=1;@i<=nrows(#[1]);@i=@i+1)
    {
      @c=@c+#[1][@i,1]*gen(@i);
    };
  return(@c);
};

proc falfa(list #)
Prameters (int i,matrix xi2, matrix d1,matrix d2).
Given a coblundary xi in Hom^2(L.,L.) with
xi2:L_2--->L_0, this procedure returns an element ,
alfa( 1 or 2 ) in Hom^1(L.,L.) mapping to xi2. See
the paper following this library.
{
  if(#[1]==1)
  {return(falfa1(#[2],#[3],#[4]));}
  else
  { return(falfa2(#[2],#[3],#[4]));};
};

proc ifind(list #)
Parameters (poly m, poly m1, ideal BB)
returns the integer i such that m1*BB(i)=m if possible.
else 0.
{
  int @v;
  for (int @i=1;@i<=ncols(#[3]);@i=@i+1)
  {
    if(#[3][@i]*#[2]==#[1])
    {
      @v=@i;
    };
  };
  return(@v);
};

proc zeromatrix(list #)
{
  matrix @f[#[1]][#[2]];
  return(@f);
};
```

```
proc augmentker(list #)
This procedure returns the matrix of the augmentation of the kernel,that is the
module in the "Singular" sence
{
return(modulo(#[2],#[3]));
};



proc quotkerim(list #)
This procedure returns the module-representation of the kernel of a mapping
divided on the image of a mapping:
R^n/P1--a-->R^m/P2--b-->R^s/P3
{
module @e=#[1];
module @sum=#[2]+@e;
module @aug=augmentker(#[2],#[3],#[4]);
return(modulo(@aug,@sum));
};

proc freerep(list #)
given a morphism M^m--->M^n as a n x m- matrix with entries in the basering. Assuming that
M=R^p/P, the procedure returns the corresponding (np x mp)-matrix representing the morphism.
{
int @p;
int @n;
int @m;
int @j;
int @k;
@p=nrows(#[1]);
@n=nrows(#[2]);
@m=ncols(#[2]);
matrix @F[@n*@p][@m*@p];
for(int @i=1;@i<=@n;@i=@i+1)
{ for( @j=1;@j<=@m;@j=@j+1)
{ for( @k=1;@k<=@p;@k=@k+1)
{ @F[@k+(@i-1)*@p,@k+(@j-1)*@p]=#[2][@i,@j];};};};
return(@F);
};

proc moduleexp(list #)
This procedure returns the module M to the n`th.
{
int @p=nrows(#[1]);int @r=ncols(#[1]);
int @k;int @i;int @j;
matrix @F[@p*#[2]][@r*#[2]];
for (@k=0;@k<=#[2]-1;@k=@k+1)
{for (@i=1;@i<=@p;@i=@i+1)
{for (@j=1;@j<=@r;@j=@j+1)
{@F[@i+@p*@k,@j+@r*@k]=#[1][@i,@j];};};};
return(@F);
};


proc matrixofmodule(list #)
```

```
{
matrix @du[nrows(#[1])][ncols(#[1])]=#[1];
return(@du);};



proc basisExt(list #)
Input parameters: i,M,N,<2,...,i+1`th differentials in the resolution of M>.
As indicated, the last parameter is optional,
and can be filled in if you want to use a particular
resolution when computing Ext^i(M,N). Note that the first differential
is already defined.
{
 matrix @d(1..(#[1]+1));@d(1)=matrixofmodule(#[2]);
 if (size(#)==#[1]+3)
 {
  for(int @k=1;@k<=#[1];@k=@k+1)
  {
    @d(@k+1)=#[@k+3];
  }
 }
 else
 {
  res(#[2],#[1]+1,@v);
  for(int @i=2;@i<=#[1]+1;@i=@i+1)
  {
   @d(@i)=matrixofmodule(@v(@i));
   };
 };
 module @m=moduleexp(#[3],ncols(@d(#[1])));
 module @n=moduleexp(#[3],ncols(@d(#[1]+1)));
 matrix @a=transpose(@d(#[1]));
 matrix @b=transpose(@d(#[1]+1));
 module @q=quotkerim(freerep(#[3],@a),@m,freerep(#[3],@b),@n);
 module @v=kbase(std(@q));
 module @o=augmentker(@m,freerep(#[3],@b),@n);
 int @l=nrows(#[3])*ncols(@d(#[1]));
 int @j=ncols(@v);
 int @t=nrows(@v);
 matrix @r1[@l][@j];
 matrix @v1=matrixofmodule(@v);
 matrix @o1=matrixofmodule(@o);
 matrix @v2[@t][1];
 matrix @c[1][1];
 for (int @k1=1;@k1<=@j;@k1=@k1+1)
 {
   @v2=@v1[1..@t,@k1];
   if (@l==1)
   {
     @c=@o1*@v2;
     @r1[1..@l,@k1]=@c[1,1];
   }
   else
   {
     @r1[1..@l,@k1]=@o1*@v2;
   };

 }
```

```
 return(@r1);

};

proc summatrix
parameters (matrix m1,matrix m2)
{  //"summerer";print(#[1]);print(#[2]);
  matrix @m2[nrows(#[1])][ncols(#[1])];
  if (iszero(#[1])==0)
  {return(#[2]);}
  else
  {
    @m2=#[2];
    if(iszero(@m2)==0)
    {
      @m2=zeromatrix(nrows(#[1]),ncols(#[1]));
    };
    return(#[1]+@m2);};
  };
};

proc iszero
{
 int @i;int @j;int @t;
 for(@i=1;@i<=nrows(#[1]);@i=@i+1)
 {
   for(@j=1;@j<=ncols(#[1]);@j=@j+1)
   {
    if(#[1][@i,@j]!=0)
    {@t=1;};
   };
  };
  return(@t);
};

proc matrixofvector
parameters (a,i,j).
{
 int @i;int @j;matrix @a1[#[2]][#[3]];
 for (@j=1;@j<=#[3];@j=@j+1)
 {
 for (@i=1;@i<=#[2];@i=@i+1)
 {@a1[@i,@j]=#[1][@i+(@j-1)*#[2],1];};
 };
 return(@a1);
};

proc alfa(list #)
Takes as input (int i,n x 1-matrix a, int p,module M,<d2,...,di+1>).
a is a column vector in Ext^p(M,M), and the procedure returns the
i`th morphism in the correspondence Ext^p(M,M)=H^p(Hom^.(L.,L.)).
{
  if (#[1]<#[3]){return(0)} else
  {
    matrix @d(1..#[1]);
    @d(1)=matrixofmodule(#[4]);
    if (size(#)==#[1]+3)
    {
```

```
      for (int @k=2;@k<=#[1];@k=@k+1)
      {
        @d(@k)=#[@k+3];
      }
    }
    else
    {
      res(#[4],#[1],@v);
      for (int @k1=2;@k1<=#[1];@k1=@k1+1)
      {
        @d(@k1)=matrixofmodule(@v(@k1));
      };
    };
    int @i1=nrows(@d(1));int @j1=ncols(@d(#[3]));
    matrix @a0=matrixofvector(#[2],@i1,@j1);
    for (int @i3=#[3];@i3<=#[1]-1;@i3=@i3+1)
    {
      @a0=((-1)^#[3])*lift(@d(@i3-#[3]+1),@a0*@d(@i3+1));
    };
    return(@a0);
  };
};



proc basisfrom(int i,ideal q,ideal SC);
{
  ideal q1,r1;ideal sp;
  q1=maxideal(i+1)+q;q1=std(q1);
  sp=remove2(SC,maxideal(i));
  r1=reduce(sp,q1);
  r1=remove2(r1,sp);
  q1=r1,SC;q1=removezeros(q1);
  return(q1);
};


proc relyp(module B,module BC,module Q)
Assumes that B is a basis for k[u]/Q and returns the matrix
R(i,) such that B(i)^*(BC(j))=R(i,j). Also assumes that the basering is
S (see proc moduli).
{
  matrix r1[ncols(B)][ncols(BC)+1];
  int n1=nrows(r1);int n2=ncols(r1);int i;
  module BQ=B,Q;
  module M=modulo(BQ,Q);
  matrix I[n1+ncols(Q)][n1];
  matrix free=freemodule(n1);
  I[1..n1,1..n1]=free;
  matrix F=lift_kbase(I,M);
  F=lift(F,freemodule(n1));
  module psi;matrix phi;
  for(i=1;i<=n1;i=i+1)
  {
        r1[i,1]=1;
  };
  for(i=2;i<=n2;i=i+1)
```

```
  {
        psi=BC[i-1];
        psi=lift(BQ,psi);
        phi=lift_kbase(psi,M);
        r1[1..n1,i]=F*phi;
  };
  return(r1);
}


proc getideal(list #)
parameters (ideal B, int from, int to)
{
  ideal @B;
  for(int @i=#[2];@i<=#[3];@i=@i+1)
  {
    @B=@B+#[1][@i];
  };
  return(@B);
};

proc findi(list #)
Parameters (n,i). Returns the index j such that i[j]=n.
{
  for (int @i=1;@i<=ncols(#[2]);@i=@i+1)
  {
    if(#[2][@i]==#[1])
    {
      return(@i);
    };
  };
  return(0);
};

proc printmatrix(matrix m)
{  "Matrix of size";
  int i,j;nrows(m)*10+ncols(m);
  "Read row by row";
  for(i=1;i<=nrows(m);i=i+1)
  {
   for(j=1;j<=ncols(m);j=j+1)
   {
    print(m[i,j]);
   };
  };
};

proc reducematrix(matrix m);
{
  int i,j;matrix rm[nrows(m)][ncols(m)];
  ideal null=std(0);
  for(i=1;i<=nrows(m);i=i+1)
  {
   for(j=1;j<=ncols(m);j=j+1)
   {
    rm[i,j]=reduce(m[i,j],null);
   };
  };
```

```
  return(rm);
};

proc moduli(list #)
Defines a ring S and matrices alf1,alf2. Returns the ideal in S defining
the relations for the formal moduli. The defining systems are recorded
in alf, over the defined bases BB.Parameters (module m,int nrl,<d2,d3>).
{  "Version 7";
  if(size(#)==4)
  {
    matrix @d2[nrows(#[3])][ncols(#[3])]=#[3];
    matrix @d3[nrows(#[4])][ncols(#[4])]=#[4];
  }
  else
  {
    res(#[1],3,@v);
    matrix @d2[nrows(@v(2))][ncols(@v(2))]=@v(2);
    matrix @d3[nrows(@v(3))][ncols(@v(3))]=@v(3);
  };
  matrix @d1=matrixofmodule(#[1]);
  matrix @xb=basisExt(1,#[1],#[1],@d2);
  matrix @yb=basisExt(2,#[1],#[1],@d2,@d3);
  module ybc;
  int e1=ncols(@xb);int @e2=ncols(@yb);
  ring S=0,u(1..e1),Dp;
  ideal m=u(1..e1);ideal ma=kbase(std(m^2));
  setring a;
  matrix alf1(1..e1+1);matrix alf2(1..e1+1);
  matrix @c1[nrows(@xb)][1];
  alf1(1)=@d1;alf2(1)=@d2;
  for(int @i=1;@i<=e1;@i=@i+1)
  {
    @c1=@xb[1..nrows(@xb),@i];
    alf1(@i+1)=alfa(1,@c1,1,#[1]);
    alf2(@i+1)=alfa(2,@c1,1,#[1],@d2);
  };
 //SOME DECLARATIONS OVER RING a:
  int @j,@i1,@i2,@i3,@i4,@m1,@m2,@iBBC;int @bp(1)=0;int @iy(1)=0;
  matrix @w1,@w2,@w3,@minusbm,Extrel;int nrl=#[2];
  poly @befa;module @d2t;
  map @kappa=S,0;
 //SOME DECLARATIONS OVER RING S:
  setring S;
  ideal frif,@BB,@BP,@B,@BC,@BBC,@BPC,@BBPC,@my,@rel,colid,TT;
  module base,M,frifmodule,T1,T2,T3;
  matrix primerel,basematrix,frifmatrix;
  poly @betap,@n1,@n2,@t,f(1..@e2);
  map @slappa=a,0;
  while (@j<nrl)
 {  "LOOP";@j+1;
    @j=@j+1;TT=basisfrom(@j+1,intersect(m^(@j+1),m*frif),m*@BC);
    @BPC=sumideal(@BPC,TT);
    int @bp(@j+1);
    if(@BPC[1]!=0)
    {
       @bp(@j+1)=ncols(@BPC);
     };
    @BBPC=sumideal(@BBC,TT);
```

```
if(@bp(@j+1)>=@bp(@j)+1)
{
  @BP=remove2(getideal(@BPC,@bp(@j)+1,@bp(@j+1)),m^(@j+1));
}
else
{
  @BP=m^(@j+1);
};
M=modulo(frif,maxideal(1)*frif);
base=kbase(std(M));
basematrix=base;
frifmodule=frif;
frifmatrix=frifmodule;
basematrix=frifmatrix*basematrix;
base=basematrix;
@BB=remove2(@BBC,kbase(std(maxideal(@j+1))));
if (base[1]!=0)
{
base=@BP,@BB,base;
}
else
{
base=@BP,@BB;
};
T1=base;T2=@BBPC;T3=maxideal(1)*frif+maxideal(@j+2);
primerel=relyp(T1,T2,T3);
colid=0;
if ( @BBPC[1]!=0)
{
  colid=colid,@BBPC;
};
for (@i=1;@i<=ncols(@BP);@i=@i+1)
{
 colid[1]=@BP[@i];
 setring a;
 @w1=0;
 setring S;
     for(@i1=1;@i1<=ncols(colid);@i1=@i1+1)
     {
   if(primerel[@i,@i1]!=0)
   {
     @betap=primerel[@i,@i1];
         for (@i2=1;@i2<=ncols(ma);@i2=@i2+1)
     {
          @n1=ma[@i2];@m2=ifind(colid[@i1],@n1,ma);
          setring a;
          if(@m2!=0)
          {
            @w1=summatrix(@w1,@kappa(@betap)*(alf1(@i2)*alf2(@m2)));
          };
          setring S;
     };
   };
 };
     setring a;
 if (iszero(@w1) !=0)
 {
   setring S;
```

```
    if(@my[1]==0)
    {
      @m1=1;
    }
    else
    {
      @m1=ncols(@my)+1;
    };
    @my[@m1]=@BP[@i];
    setring a;
    matrix @yn(@m1)=@w1;
  };
   setring S;
};
int @iy(@j+1)=ncols(@my);
 setring a;
 ybc=[0];
 for (@i2=1;@i2<=@iy(@j+1)-@iy(@j);@i2=@i2+1)
 {
       ybc[@i2]=svectorofmatrix(vectorofmatrix(@yn(@iy(@j)+@i2)));
 };
 @d2t=freerep(#[1],transpose(@d2)),moduleexp(#[1],ncols(@d2));
Extrel=relyp(@yb,ybc,@d2t);
 setring S;
  for ( @i1=1;@i1<=@e2;@i1=@i1+1)
  {
    @t=0;
    for (@i2=1;@i2<=@iy(@j+1)-@iy(@j);@i2=@i2+1)
    {
      setring a;
      @befa=Extrel[@i1,@i2+1];
      setring S;
      @t=@t+@slappa(@befa)*@my[@iy(@j)+@i2];
    };
    f(@i1)=f(@i1)+@t;
  };
 frif=f(1..@e2);@BP=0;
 if(@bp(@j+1)>@bp(@j))
 {
   TT=getideal(@BPC,@bp(@j)+1,@bp(@j+1));
   @BC=basisfrom(@j+1,intersect(frif,maxideal(@j+1)),TT);
 }
 else
 {
   @BC=basisfrom(@j+1,intersect(frif,maxideal(@j+1)),0);
 };
 @BBC=sumideal(@BBC,@BC);
 @iBBC=ncols(@BBC);
 @B=remove2(@BC,m^(@j+1));
 base=@B,@BB;
 T1=base;T2=@BBC;T3=frif+maxideal(@j+2);
 primerel=relyp(T1,T2,T3);
 colid=0;
 if ( @BBC[1]!=0)
 {
  colid=colid,@BBC;
 };
 for(@i1=1;@i1<=ncols(@B);@i1=@i1+1)
```

```
    {
      setring a;
      @minusbm=0;
      setring S;
      colid[1]=@B[@i1];
      for(@i3=1;@i3<=ncols(colid);@i3=@i3+1)
      {
        @betap=primerel[@i1,@i3];
        if(@betap!=0)
        {
          @i2=findi(colid[@i3],@my);
          setring a;
          if(@i2!=0)
          {
            @minusbm=summatrix(@minusbm,@kappa(@betap)*@yn(@i2));
          };
        };
        setring S;
      };
      setring a;
      @minusbm=-1*@minusbm;
      if(iszero(@minusbm)!=0)
      {
        setring S;
        @i4=ncols(ma);
        ma[@i4+1]=@B[@i1];@i4=@i4+1;setring a;
        matrix alf1(@i4)=falfa(1,@minusbm,@d1,@d2);
        matrix alf2(@i4)=falfa(2,@minusbm,@d1,@d2);
      };
      setring S;
    }

  };
 "THE LOCAL MODULI IS THE FREE POWER SERIES RING OF DIMENSION";
 e1;
 "MODULO THE IDEAL GENERATED BY THE FOLOWING POLYNOMIALS:";
 print(frif);"THE VERSAL FAMILY IS GIVEN BY";
//Here we make the versal family
 ring ah=0,(x,y,u(1..e1)),Dp;
 map atoah=a,(x,y);
 map Stoah=S,u(1..e1);
 ideal frif=Stoah(frif);
 ideal q=x4+y3;
 ideal qv=q,frif;
 qring AtH=std(qv);
 map atoath=a,(x,y);
 map Stoath=S,u(1..e1);
 matrix stepvers1[nrows(atoath(@d1))][ncols(atoath(@d1))];
 matrix stepvers2[nrows(atoath(@d2))][ncols(atoath(@d2))];
 int w1,w2,w3;
 for(w1=1;w1<=ncols(Stoath(ma));w1=w1+1)
 {
   for(w2=1;w2<=ncols(stepvers1);w2=w2+1)
   {
     for(w3=1;w3<=ncols(stepvers1);w3=w3+1)
     {
      stepvers1[w2,w3]=stepvers1[w2,w3]+atoath(alf1(w1))[w2,w3]*Stoath(ma)[w1];
     };
```

```
    };
  };
  for(w1=1;w1<=ncols(Stoath(ma));w1=w1+1)
  {
    for(w2=1;w2<=ncols(stepvers2);w2=w2+1)
    {
      for(w3=1;w3<=ncols(stepvers2);w3=w3+1)
      {
        stepvers2[w2,w3]=stepvers2[w2,w3]+atoath(alf2(w1))[w2,w3]*Stoath(ma)[w1];
      };
    };
  };
  printmatrix(stepvers1);
  "It is versal if the following matrix is zero";
  matrix obst=stepvers1*stepvers2;
  obst=reducematrix(obst);
  print(obst);

};
```

References

La 1. Matric Massey products and formal moduli I, Lecture Notes in Mathematics, Springer-Verlag, No 1183 (1986).pp.218-240.

La 2: O. A. Laudal. Formal moduli of algebraic structures, Lecture Notes in Mathematics, Springer-Verlag, No.754 (1978).

Si: The program Singular

Siq 1: A. Siqveland. Thesis, University of Oslo (1996).