NCPOLY: Computation in non–commutative polynomial ideals

Herbert Melenk
Konrad–Zuse–Zentrum für Informationstechnik Berlin
Takustrasse 7
D–14195 Berlin – Dahlem
Germany
E–mail: melenk@zib.de

Joachim Apel
Institut für Informatik
Universität Leipzig
Augustusplatz 10–11
D–04109 Leipzig
Germany
E–mail: apel@informatik.uni–leipzig.de

1 Introduction

REDUCE supports a very general mechanism for computing with objects under a non–commutative multiplication, where commutator relations must be introduced explicitly by rule sets when needed. The package **NCPOLY** allows you to set up automatically a consistent environment for computing in an algebra where the non–commutativity is defined by Lie-bracket commutators. The package uses the REDUCE **noncom** mechanism for elementary polynomial arithmetic; the commutator rules are automatically computed from the Lie brackets. You can perform polynomial arithmetic directly, including **division** and **factorization**. Additionally **NCPOLY** supports computations in a one sided ideal (left or right), especially one

sided **Gröbner** bases and **polynomial reduction**.

2 Setup, Cleanup

Before the computations can start the environment for a non–commutative computation must be defined by a call to **nc_setup**:

```
nc_setup(<vars>[,<comms>][,<dir>]);
```

where

$< vars >$ is a list of variables; these must include the non–commutative quantities.

$< comms >$ is a list of equations `<u>*<v> - <v>*<u>=<rh>` where $< u >$ and $< v >$ are members of $< vars >$, and $< rh >$ is a polynomial.

$< dir >$ is either $left$ or $right$ selecting a left or a right one sided ideal. The initial direction is $left$.

nc_setup generates from $< comms >$ the necessary rules to support an algebra where all monomials are ordered corresponding to the given variable sequence. All pairs of variables which are not explicitly covered in the commutator set are considered as commutative and the corresponding rules are also activated.

The second parameter in **nc_setup** may be omitted if the operator is called for the second time, e.g. with a reordered variable sequence. In such a case the last commutator set is used again.

Remarks:

- The variables need not be declared **noncom** - **nc_setup** performs all necessary declarations.
- The variables need not be formal operator expressions; **nc_setup** encapsulates a variable x internally as `nc!*(!_x)` expressions anyway where the operator $nc!*$ keeps the noncom property.
- The commands **order** and **korder** should be avoided because **nc_setup** sets these such that the computation results are printed in the correct term order.

Example:

```
nc_setup({KK,NN,k,n},
           {NN*n-n*NN= NN, KK*k-k*KK= KK});

NN*n;             ->   NN*n
n*NN;             ->   NN*n - NN
nc_setup({k,n,KK,NN});
NN*n - NN         ->   n*NN;
```

Here KK, NN, k, n are non–commutative variables where the commutators are described as $[NN, n] = NN$, $[KK, k] = KK$.

The current term order must be compatible with the commutators: the product $< u > * < v >$ must precede all terms on the right hand side $< rh >$ under the current term order. Consequently

- the maximal degree of $< u >$ or $< v >$ in $< rh >$ is 1,
- in a total degree ordering the total degree of $< rh >$ may be not higher than 1,
- in an elimination degree order (e.g. lex) all variables in $< rh >$ must be below the minimum of $< u >$ and $< v >$.
- If $< rh >$ does not contain any variables or has at most $< u >$ or $< v >$, any term order can be selected.

If you want to use the non–commutative variables or results from non–commutative computations later in commutative operations it might be necessary to switch off the non–commutative evaluation mode because not all operators in REDUCE are prepared for that environment. In such a case use the command

```
nc_cleanup;
```

without parameters. It removes all internal rules and definitions which **nc_setup** had introduced. To reactive non–commutative call **nc_setup** again.

3 Left and right ideals

A (polynomial) left ideal L is defined by the axioms

$u \in L, v \in L \Longrightarrow u + v \in L$

$u \in L \Longrightarrow k * u \in L$ for an arbitrary polynomial k

where "*" is the non–commutative multiplication. Correspondingly, a right ideal R is defined by

$u \in R, v \in R \Longrightarrow u + v \in R$

$u \in R \Longrightarrow u * k \in R$ for an arbitrary polynomial k

4 Gröbner bases

When a non–commutative environment has been set up by **nc_setup**, a basis for a left or right polynomial ideal can be transformed into a Gröbner basis by the operator **nc_groebner**:

```
nc_groebner(<plist>);
```

Note that the variable set and variable sequence must be defined before in the **nc_setup** call. The term order for the Gröbner calculation can be set by using the **torder** declaration. The internal steps of the Gröbner calculation can be watched by setting the switches **trgroeb** (=list all internal basis polynomials) or **trgroebs** (=list additionally the S-polynomials) [1].

For details about **torder**, **trgroeb** and **trgroebs** see the **REDUCE GROEBNER** manual.

```
2: nc_setup({k,n,NN,KK},{NN*n-n*NN=NN,KK*k-k*KK=KK},left);

3: p1 := (n-k+1)*NN - (n+1);

p1 :=  - k*nn + n*nn - n + nn - 1

4: p2 := (k+1)*KK -(n-k);

p2 := k*kk + k - n + kk

5: nc_groebner ({p1,p2});
```

[1] The command `lisp(!*trgroebfull:=t);+ causes additionally all elementary polynomial operations to be printed.`

```
{k*nn - n*nn + n - nn + 1,

 k*kk + k - n + kk,

 n*nn*kk - n*kk - n + nn*kk - kk - 1}
```

Important: Do not use the operators of the GROEBNER package directly as they would not consider the non–commutative multiplication.

5 Left or right polynomial division

The operator **nc_divide** computes the one sided quotient and remainder of two polynomials:

```
nc_divide(<p1>,<p2>);
```

The result is a list with quotient and remainder. The division is performed as a pseudo–division, multiplying $< p1 >$ by coefficients if necessary. The result $\{< q >, < r >\}$ is defined by the relation

$< c > * < p1 >=< q > * < p2 > + < r >$ for direction $left$ and

$< c > * < p1 >=< p2 > * < q > + < r >$ for direction $right$,

where $< c >$ is an expression that does not contain any of the ideal variables, and the leading term of $< r >$ is lower than the leading term of $< p2 >$ according to the actual term order.

6 Left or right polynomial reduction

For the computation of the one sided remainder of a polynomial modulo a given set of other polynomials the operator **nc_preduce** may be used:

```
nc_preduce(<polynomial>,<plist>);
```

The result of the reduction is unique (canonical) if and only if $< plist >$ is a one sided Gröbner basis. Then the computation is at the same time an ideal membership test: if the result is zero, the polynomial is member of the ideal, otherwise not.

7 Factorization

Polynomials in a non–commutative ring cannot be factored using the ordinary **factorize** command of REDUCE. Instead one of the operators of this section must be used:

```
nc_factorize(<polynomial>);
```

The result is a list of factors of $<$ *polynomial* $>$. A list with the input expression is returned if it is irreducible.

As non–commutative factorization is not unique, there is an additional operator which computes all possible factorizations

```
nc_factorize_all(<polynomial>);
```

The result is a list of factor decompositions of $<$ *polynomial* $>$. If there are no factors at all the result list has only one member which is a list containing the input polynomial.

In contrast to factoring in commutative polynomial rings, the non–commutative factorization is rather time consuming. Therefore two additional operators allow you to reduce the amount of computing time when you look only for isolated factors in special context, e.g. factors with a limited degree or factors which contain only explicitly specified variables:

```
 left_factor(<polynomial>[,<deg>[,<vars>]])
right_factor(<polynomial>[,<deg>[,<vars>]])
 left_factors(<polynomial>[,<deg>[,<vars>]])
right_factors(<polynomial>[,<deg>[,<vars>]])
```

where $<$ *polynomial* $>$ is the form under investigation, $<$ *vars* $>$ is an optional list of variables which must appear in the factor, and $<$ *deg* $>$ is an optional integer degree bound for the total degree of the factor, a zero for an unbounded search, or a monomial (product of powers of the variables) where each exponent is an individual degree bound for its base variable; unmentioned variables are allowed in arbitrary degree. The operators $*\_factor$ stop when they have found one factor, while the operators $*\_factors$ select all one–sided factors within the given range. If there is no factor of the desired type, an empty list is returned by $*\_factors$ while the routines $*\_factor$ return the input polynomial.

The share variable *nc_factor_time* sets an upper limit for the time to be spent for a call to the non–commutative factorizer. If the value is a positive integer, a factorization is terminated with an error message as soon as the time limit is reached. The time units are milliseconds.

8 Output of expressions

It is often desirable to have the commutative parts (coefficients) in a non–commutative operation condensed by factorization. The operator

```
nc_compact(<polynomial>)
```

collects the coefficients to the powers of the lowest possible non-commutative variable.

```
load ncpoly;

nc_setup({n,NN},{NN*n-n*NN=NN})$
p1 := n**4 + n**2*nn + 4*n**2 + 4*n*nn + 4*nn + 4;

       4    2         2
p1 := n  + n *nn + 4*n  + 4*n*nn + 4*nn + 4

nc_compact p1;

  2     2          2
(n  + 2)  + (n + 2) *nn
```