Supplying involutive bases

Klaus-Peter Nischke

Deliverable 3.3.1.2.1

1 Introduction

This report presents the integration of the involutive algorithm for computing Gröbner bases into the PoSSo library.

The involutive algorithm is a special form of the Buchberger algorithm and in many cases provides better convergence and numerical stability than the classical form.

The main idea of the involutive algorithm is to disallow the reduction of some power products by some basis elements; this is done through a special structure (involutive division) defining "multiplicative" and "non-multiplicative" variables for a given power-product; this is used instead of the "usual" divisibility test in Buchberger algorithm to decide if a power-product can be simplified by a simplifier. The result of the involutive algorithm is a (possibly redundant) Gröbner basis. Though some of the involutive divisions might fail to give a convergent algorithm (the involutive basis might be infinite), the present implementation is designed to terminate and stop with a Gröbner basis for every degree-compatible ordering and arbitrary involutive division. Special issues of involutive division are the Janet division, the Pommaret division, the Thomas division. The first two guarantee convergence, the third one only if the ideal is zero-dimensional.

Preliminary implementations have shown that the involutive algorithm has a better behaviour than the usual Buchberger algorithm in many cases, and it is expected that the algorithm or some variant will be much more suitable for computations with floating coefficients. The experiments will be conducted in the future, and will be the object of a future report.

For a fuller account of the algorithm see [1] and [2].

The integration was designed to extend the existing PoSSo library as smooth as possible.

The implementation has been done in the release 2.0.4, the source code can be found in the lib/Groebner directory in the invbase{.C,.H} and invdiv{.C,.H} files. The documentation has been included in the PoSSo library Reference Manual, plgroeb.texi file.

2 Software Design Decisions

The involutive algorithm is implemented as a c++ class `PL_InvolutiveReductor` and may be used as a drop-in replacement of `PL_GroebnerReductor`. Since much of the functionality of `PL_GroebnerReductor` could be reused or almost reused by `PL_InvolutiveReductor`, the latter was derived from `PL_GroebnerReductor`. The possible customizations provided by `PL_GroebnerReductor` were utilized and new customizations added as needed.

3 Description of New Code

3.1 Involutive Divisions

Both general involutive divisions and the special division principles of *Janet*, *Pommaret* and *Thomas* are implemented within the module `invdiv`.

`class InvDivision` implements a general involutive division by associating multiplicative variables $M(s)$ to terms $s \in S$, S a finite set of terms.

The main work is done by `class InvDiv_Implementation` hiding the implementation (so it is not visible outside of `invdiv.C`) using an open hashing table.

`class InvolutiveDivision` is an abstract base for a given division principle. Its intention is to define a member for recomputing all multiplicative variables.

The derived `class JanetDivision`, `class PommaretDivision` and `class ThomasDivision` implement these specific division principles.

3.2 New Customizations of `PL_GroebnerReductor`

- `PL_Bool (*DoesDivide) (`
  `PL_GroebnerReductor &,const PL_PP &s,const PL_PP &t);`
  Returns `TRUE`, if `s` divides `t` in some sense (i.e. the conventional division in the Buchberger case, and the involutive division when using `PL_InvolutiveReductor`).

- `void (*PreProcessPairHook) (`
  `PL_GroebnerReductor &, PL_Pair *);`
  Will be called prior to the handling of a new critical pair. In the Buchberger case, this function is a no-op; in the involutive case it introduces some additional simplifiers.

3.3 Involutive Algorithm

As mentioned above, the involutive algorithm utilizes much of the functionality provided by the existing class `PL_GroebnerReductor`. To interface with this, the following members of class `PL_InvolutiveReductor` are provided:

- `static void InvolutiveSimplifiersAppend(`
  `PL_GroebnerReductor &, PL_Poly *);`
  Called to append a new simplifier. For this new polynomial, the involutive division has to be updated (enlarged by the leading term), and the multiplicative variables to be recomputed for all simplifiers.

- `void AutoreduceSimplifiers(const PL_PP &UpperLimit);`
  Autoreduces the list of all simplifiers with leading term being less than or equal to `UpperLimit`.

- `void UpdateInvDivision(const PL_PP &UpperLimit);`
  Update the list of simplifiers so that their leading terms define an involutive division being complete up to `UpperLimit`.

- `static void InvolutivePreprocessPairHook(`
  `PL_GroebnerReductor &, PL_Pair *c);`
  Called prior to processing a new pair. This function will introduce new simplifiers until the involutive division is complete up to the leading term of `c`, and (optionally) reduce or autoreduce the new simplifiers.

- `static PL_Bool InvolutiveDoesDivide(`
  `PL_GroebnerReductor &,const PL_PP &s, const PL_PP &t);`
  Called to determine, whether `t` involutively divides `s`. That is, `t` divides `s`, and `s/t` is a product of the multiplicative variables for `t`.

- `static PL_PolyList &InvolutiveFinalReduction(`
  `PL_GroebnerReductor &);`
  Called to compute the final reduction. First this function will sort out all simplifiers with the leading term being divisible by the leading term of some other simplifier. Finally, depending on the choosen option for reducing intermediate simplifiers, this function will totally autoreduce all simplifiers or just return.

Public Interface

- `PL_InvolutiveReductor( PL_InvolutiveReductor &ir )`
  Copy-constructor.

- `PL_InvolutiveReductor(PL_PolyList *InSimplifiers = NULL,`
```
    PL_PolyList *InBasis = NULL,
    PL_List<PL_Pair> *InPairs = NULL ,
    PL_PolyRing *InPolyRing = NULL):
    PL_GroebnerReductor(
        InSimplifiers,InBasis,InPairs,InPolyRing)
```
  The "normal" constructor.

- `PL_PolyList &InvComputeBasis`
```
    (const InvolutiveDivision &DivisionMethod=JanetDivision,
    PL_PolyList &InitialGenerators = EmptyList,
    InvReductionMode Mode=InvReduceNew);
```
  Compute the groebner basis of the ideal given my `InitialGenerators` using the involutive algorithm. This function operates by setting the appropriate function pointers for `PL_GroebnerReductor` and then calling `PL_GroebnerReductor`'s function `ComputeBasis(...)`.

  The first parameter chooses the division principle (those currently implemented are `JanetDivision`, `PommaretDivision` and `ThomasDivision`, please see above).

  The third parameter chooses a strategy for tail-reducing simplifiers. Allowed values are

  – `InvAutoreduce` Before each computation of S-polynomials, all simplifiers up to the leading term of this S-polynomial are fully autoreduced.

  – `InvReduceNew` Whenever a new intermediate simplifier is added, this one is reduced using the already existing ones.

  – `InvReduceNever` New simplifiers are never tail-reduced.

4 Integration

Support for the involutive algorithm has been added to the PoSSo programs `SampleSolver` and `PoSSoServer`.

For the `SampleSolver` the flag `I` has been added.

If present, the computation will use the involutive algorithm instead of the 'conventional' one. The division principle and the reduction strategy are optional argument.

The user can choose from `-I`, `-Is` or `-Is,r` to select the involutive algorithm with:

- `s=0` Janet division (default)
- `s=1` Pommaret division
- `s=2` Thomas division
- `r=0` reduce new simplifiers (default)
- `r=1` never tail-reduce intermediate simplifiers
- `r=2` always keep simplifiers autoreduced.

In the `PoSSoServer` two commands have been added:

- XCL_Values GServer::IR_Open(XCL_Channel & s)

  The PL_InvolutiveReductor is created instead of a PL_GroebnerReductor for computing the basis.

- XCL_Values GServer::IR_ComputeBasis(XCL_Channel & s)

  It runs the involutive algorithm on the PL_InvolutiveReductor with the JanetDivision. The other kind of divisions, Pommaret and Thomas, will be integrated in a future work.

  Here is an example program showing how to use the involutive algorithm with the PoSSoServer.

```
Nop
%       PoSSo Server Test
%
%       Various Involutive ComputeBasis
%
@

Clear@
SetAlgoVerbosity 1
@
SetResultVerbosity 2
@
PR_Open
{ {0},
  {2,{4,
        {N, {x, y, z, t}},
        {C, {1, 1, 1, 1}},
        {V, {1, 1, 1, 1}}
  }}
}                                       % open PR(0)
@
PR_SetCurrent (0)
@
PL_Open (0)                             % open PL(1)
@
PL_Put (1) {{1x,1y,1z,1t},{1xy,1yz,1zt,1tx},{1xyz,1yzt,1ztx,1tyx},{1xyzt,-1}}
@
IR_Open (0)
@
IR_ComputeBasis (2) (1)
@
GR_Get (2)                              % getting GR returns
                                        % its basis
@
IR_Open (0)                             % open IR(3)
@
PL_Put (1)
{{1t2,-1t,2z2,2y2,2x2}
,{2tz,2zy,2yx,-1z}
,{2ty,1z2,2zx,-1y}
,{1t,2z,2y,2x,-1}
}
@
IR_ComputeBasis (3) (1)
@
```

```
GR_Get (3)
@
IR_Open (0)
                                                    % open IR(4)
@
PL_Put (1)
{{6x4y3,21x4y2z,15x4yz2,9x4z3,-8x2y2t,-28x2yzt,36x2z2t,-144x2y,-648x2z,-120}
,{9x4y4,30x4y3z,39x4y2z2,18x4yz3,-24x2y3t,-16x2y2zt,16x2yz2t,24x2z3t,
-432x2y2,-720x2yz,-432x2z2,16y2t2,-32yzt2,16z2t2,576yt,-576zt,-240y,5184}
,{-15x2y3t,15x2y2zt,-81x2y2,216x2yz,-162x2z2,40y2t2,-80yzt2,40z2t2,1008yt,
-1008zt,5184}
,{-4x2y2,4x2yz,-3x2z2,22yt,-22zt,261}
}
@
IR_ComputeBasis (4) (1)
@
GR_GetHead (4)
&0,{{1yz},{1y2},{1x2},{1t3},{1zt2},{1yt2},{1z2t},{1z3}} &
@
Clear@
Nop
End of groebner_1.inv.xcl
@
```

References

[1] V. P. Gerdt, Y. A. Blinkov, *Involutive Base of Polynomial Ideals*, Preprint 01/96, Naturwissenschaftlich-Theoretisches Zentrum, University of Leipzig.

[2] A. Yu. Zharkov, Yu. A. Blinkov, *Involution Approach to Solving Systems of Algebraic Equations*, IMACS SC-93, LILLE FRANCE.