CoCoALib - Feature #665

Integrate Janet/Pommaret basis code

11 Feb 2015 17:42 - John Abbott

Status:	In Progress	**Start date:**	11 Feb 2015
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	10%
Category:	New Function	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	1.60 hour

Description

Integrating Janet/Pommaret basis into CoCoALib and CoCoA-5

What Mario will do:

- think about good names for JBMill, PBMill, etc.
- JBMill exists and essentially finished.
- PBMill needs to be written; it will derive publicly from JBMill.
- main purpose of JBMill/PBMill is compute the basis (nothing more).
- Universal JB/PB container. PURPOSE: computing values once JB/PB has been computed.
- modify SparsePolyRingBase::IdealImpl to contain (auto?) ptr to universal container.
- add copyright headers to all source code (.C and .H).

- will put all his code in namespace CoCoA::involutive.
- try putting all his code in a subdir (with suitable Makefile).

What John+Anna will do:

- implement GBMill; main purpose is to compute G bases (also NF?).
- define GB container: main purpose is to compute values one a GB has been computed.
- modify SparsePolyRingBase::IdealImpl to contain (auto?) ptr to GB container.
- make several new ideal fns available in CoCoA-5.

- implement "log" version of small non-prime finite fields.

Related issues:

Related to CoCoALib - Feature #215: Janet Bases: check and include code in Co...	**Closed**	**02 Aug 2012**
Related to CoCoA-5 - Feature #216: Janet Bases: port function into CoCoA-5	**In Progress**	**02 Aug 2012**
Related to CoCoALib - Design #455: Which sets of generators in an ideal?	**New**	**03 Mar 2014**
Related to CoCoALib - Feature #387: implement algorithm(s) for resolutions	**New**	**23 Jul 2013**
Related to CoCoALib - Feature #383: Resolution/morse: integrate Mario Albert'...	**In Progress**	**27 Jun 2013**
Related to CoCoALib - Feature #24: object files collected in one directory	**Rejected**	**08 Nov 2011**
Related to CoCoALib - Bug #833: UIBC: need include file in RingWeyl.C	**New**	**08 Dec 2015**
Related to CoCoALib - Feature #835: Make Mario's new code threadsafe	**New**	**09 Dec 2015**
Related to CoCoALib - Design #871: Redesign ideals	**New**	**26 Apr 2016**

History

#1 - 13 Feb 2015 20:41 - John Abbott

Mario will try to put all his source code in his own namespace (perhaps called involutive), and will put all his files into a subdirectory of src/AlgebraicCore/.

#2 - 11 May 2015 14:10 - John Abbott

- Target version changed from CoCoALib-0.99536 June 2015 to CoCoALib-0.99540 Feb 2016

#3 - 11 Jun 2015 08:24 - Mario Albert

In the following I try to explain how I integrated the JBMill/PBMill into SparsePolyRing::IdealImpl:

UIBC

First of all I implemented a Involutive::UniversalInvolutiveBasisContainer (UIBC). It is designed to use it with SmartPtrIRC. The UIBC should act as a wrapper between JBMill/PBMill and the IdealImpl. It is initialized with a arbitrary generating set and it will only calculate a Janet basis it is really necessary. Once a Janet basis or an invariant is computed we store it in the UIBC and every additional call only take the already computed value.

Integrate UIBC to SparsePolyRing::IdealImpl

I add the following data member to SparsePolyRing::IdealImpl

```
SmartPtrIRC<Involutive::UniversalInvolutiveBasisContainer> myInvBasisContainerPtr;
```

To use this data member I introduced several functions like:

```
const std::vector<RingElem>& SparsePolyRingBase::IdealImpl::myJanetBasis() const
{
  return myInvBasisContainerPtr->myJanetBasis();
}
```

or

```
bool SparsePolyRingBase::IdealImpl::InvIamDeltaRegular() const
{
  return myInvBasisContainerPtr->IamDeltaRegular();
}
```

To avoid naming conflicts all these methods have Inv as prefix.
To get a nice interface for the user I introduced global methods like

```
const std::vector<RingElem>& Involutive::JanetBasis(const ideal& I)
{
  if (!IsSparsePolyRing(RingOf(I)))
    CoCoA_ERROR(ERR::NotSparsePolyRing, "JanetBasis(I)");
  const SparsePolyRingBase::IdealImpl* const ptrI =
    SparsePolyRingBase::IdealImpl::ourGetPtr(I);
  return ptrI->myJanetBasis();
}
```

or

```
bool Involutive::IsDeltaRegular(const ideal& I)
{
  if (!IsSparsePolyRing(RingOf(I)))
    CoCoA_ERROR(ERR::NotSparsePolyRing, "JanetBasis(I)");
  const SparsePolyRingBase::IdealImpl* const ptrI =
    SparsePolyRingBase::IdealImpl::ourGetPtr(I);
  return ptrI->InvIamDeltaRegular();
}
```

.

To define these methods I have to declare Involutive::JanetBasis(const ideal& I) and Involutive::IsDeltaRegular(const ideal& I) as friend in SparsePolyRingBase.

A few thoughts about this solution

In principal it works, but there are some points which I do not like:

- I have to define many new friends in SparsePolyRingBase. I am not sure if this is a *clean* solution.
- The methods Inv... are only wrappers for the corresponding methods in the UIBC. Maybe it is a better solution to make the UIBC accessible for the global methods like JanetBasis and IsDeltaRegular. Then we could get rid of these methods.

#4 - 16 Jun 2015 08:25 - Mario Albert

- Assignee set to John Abbott

At the moment the UIBC uses only the standard algorithm to compute a Janet basis TQBlockLow. But it should be possible to specify which strategy the UIBC should use.
To make this functionality accessible for the user I have to implement something like the following in SparsePolyRing.H:

```
virtual void InvSetStrategy(Involutive::StrategyFlag strat) const;
{
  myInvBasisContainerPtr->setStrategy(strat);
}
```

But this do not work:

Involutive::StrategyFlag is an enum which is defined in TmpJBMill.H.
There are two possible solutions:

- Declare the enum not in TmpJBMill.H, but in SparsePolyRing.H: I do not like this solution because nobody would expect **this** enum in **this** file.
- Forward declare this enum: Not yet possible, but will be possible in C++0X ([[http://stackoverflow.com/a/72599]])

Does anyone know an alternative solution?

#5 - 08 Dec 2015 11:42 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

With Mario's help we have integrated his code into (my copy of) the CoCoA sources.

Some questions have arisen which make me wonder about how clean our design is; I'll come to these later (perhaps after re-reading what Mario has written above).

One interface question: Mario has put his code inside the (sub-)namespace CoCoA::Involutive. This make it clear that the involutive code is doing the work, and it also avoids some ambiguities: *e.g.* some functions such as IsMonomial, hilbert and IsHomog can be answered either via a Groebner basis or via an involutive basis. At the moment the interface is clumsy: the user has to write explicitly the Involutive prefix, or else has to explicitly write using namespace Involutive.

Can a nicer interface be made?

#6 - 08 Dec 2015 12:32 - John Abbott

Here is a brief "one-way discussion" about some function interfaces: I shall consider just IsMonomial, but it is a model for several other functions.

An unsophisticated user will want to simply call IsMonomial(I) and get the answer "as quickly as possible"; so it is CoCoALib's responsibility to pick the best way.

An ideal might have a bool3 flag to represent knowledge about whether the ideal is monomial or not; if the flag is set "definitely" then the answer is immediate. But remember that such flags require careful handling to work properly in a multi-threaded context!

Assuming we cannot get an immediate answer from such a flag. If I already has a G-basis or a J-basis then we can use the basis to get the answer quickly. If both bases are available, is one a better choice than the other? My guess is that a G-basis typically has fewer elements, so that would be a better choice when the ideal actually (or probably) is monomial. I have no intuition about the case when the ideal is (probably) not monomial.

If neither basis exists, at least one must be computed. Which? (and why?)

A more advanced user may want to force the use of one basis or another for the computation; how can this be done neatly?

#7 - 08 Dec 2015 13:03 - John Abbott

A simple interface is: a function which forces computation of a GBasis (without copying the result), and a function which forces computation of a JBasis (without copying the result).

Mario points out that there may be extra parameters specifying details of the algm to use for computing the JBasis; probably something similar should apply also to computation of GBasis (*e.g.* disabling/enabling various criteria).

Addendum JAA thinks it is probably worth making this simple interface (perhaps even without the extra algm parameters initially), then later we can consider making it more sophisticated based on users' demands.

#8 - 26 Apr 2016 15:08 - John Abbott

- Related to Design #871: Redesign ideals added

#9 - 16 Jun 2016 17:19 - John Abbott

- Target version changed from CoCoALib-0.99540 Feb 2016 to CoCoALib-0.99550 spring 2017

#10 - 16 Sep 2016 16:27 - John Abbott

- Target version changed from CoCoALib-0.99550 spring 2017 to CoCoALib-0.99560

#11 - 06 Nov 2017 14:00 - John Abbott

- Target version changed from CoCoALib-0.99560 to CoCoALib-0.99600

#12 - 12 Jun 2018 18:36 - John Abbott

- Target version changed from CoCoALib-0.99600 to CoCoALib-1.0