CoCoALib - Design #1744

Implement GBMill (aka Groebner Framework)

19 May 2023 15:47 - Anna Maria Bigatti

Status:	In Progress	**Start date:**	19 May 2023
Priority:	Normal	**Due date:**	
Assignee:	Anna Maria Bigatti	**% Done:**	10%
Category:	Data Structures	**Estimated time:**	40.00 hours
Target version:	CoCoALib-0.99880	**Spent time:**	2.85 hours

Description

Investigate and design about converting current "Buchberger"'s code into a GBMill, i.e. a class containing partial computation of Buchberger's algorithm.

Which operations should it offer?
How to make it accessible through CoCoA-5?

For certain:

1. constructor
2. reduction of next pair
3. extracting partial GBasis
4. number of pairs to do
5. number of polynomial in GBasis-to-be

Related issues:

Related to CoCoALib - Feature #1743: Implement Truncated GBases for homogeneo...	**New**	**19 May 2023**

History

#1 - 19 May 2023 15:48 - Anna Maria Bigatti

- Related to Feature #1743: Implement Truncated GBases for homogeneous input added

#2 - 19 May 2023 15:57 - Anna Maria Bigatti

- Description updated

#3 - 19 May 2023 22:03 - John Abbott

There should probably be a "graded" mode, and then the possibility to know whether the partial/truncated basis is complete up to some degree.
If we ever want to implement Groebner trace then it would be helpful to be able to extract the trace from a GBMill -- this probably makes sense only when the computation is complete?

#4 - 19 May 2023 22:05 - John Abbott

I would also like to be able to perform a GB computation in chunks determined by a time-out. It may suffice to have a ProceedForTime function; or is there a better interface?

#5 - 23 May 2023 16:58 - Anna Maria Bigatti

- Status changed from New to In Progress

- % Done changed from 0 to 10

The computation of a GBasis is the core computation of many others (colon, saturation, homogenization, ...).
Such GBasis computation is then in its own special context.

Moreover, it runs in its own special ring (not the ring of the input ideal): we could call the indets in the same way (currently they are called x^0...x[n-1]) is this more handy or more confusing?

#6 - 23 May 2023 19:09 - John Abbott

A possible advantage of using "funny" indet names is that it is then clear that the value does not belong to the original poly ring.
A disadvantage is that debugging could be a real headache...

One the phone I had mentioned the possibility of "exporting" an internal polynomial to the original ring. This would automatically use the expected indet names, **but** the leading PP may not be the first one printed (if the internal ring has a different term order).
An "export" function would be useful/needed when producing a truncated GB. Also I have an application (find extra gens prior to computing the homogenization) where I want to find quickly some "small" polys in the ideal *e.g.* compute a "partial" GB with a time limit, then check to see if there are any small polys. Again this would require exporting the poly... it is no importance that the LPP may not be the first.

#7 - 25 May 2023 11:31 - Anna Maria Bigatti

As a guideline: in CoCoA-4 there was the Groebner Framework:

```
instead of using one of the normal Groebner
basis-type commands (listed in the previous section), start the
computation with one of the commands,
  * GB.Start_GBasis -- start interactive Groebner basis computation
  * GB.Start_MinGens -- start interactive minimal generator calculation
  * GB.Start_Res -- start interactive resolution computation
  * GB.Start_Syz -- start interactive syzygy computation

After starting the computation, the following commands are available:
  * GB.Complete -- Complete an interactive Groebner-type calculation
  * GB.GetBettiMatrix -- returns the Betti matrix computed so far
  * GB.GetNthSyz -- returns the part of the Nth syzygy module computed so far
  * GB.GetNthSyzShifts -- shifts of the Nth syzygy module computed so far
  * GB.GetRes -- returns the resolution computed so far
  * GB.GetResLen -- returns the length of the resolution computed so far
  * GB.ResReport -- status of an interactive resolution calculation
  * GB.Stats -- status of an interactive Groebner-type calculation
  * GB.Step, GB.Steps -- take steps in an interactive Groebner-type calculation
  * ReducedGBasis -- compute a reduced Groebner basis

//=========================  EXAMPLE  =========================\\
  Use R ::= QQ[t,x,y,z];
  I := Ideal(t^3-x, t^4-y, t^5-z);
  $gb.Start_GBasis(I);  -- start the interactive framework
  I.GBasis;  -- the Groebner basis is initially empty
Null
-------------------------------
  $gb.Steps(I,1); -- a single step of the computation
  I.GBasis;
[t^3 - x]
-------------------------------
  $gb.Steps(I,4); -- 4 more steps
  I.GBasis;
[t^3 - x, -tx + y, t^2y - x^2]
-------------------------------
  $gb.Complete(I); -- complete the computation
  I.GBasis;
[t^3 - x, -tx + y, -ty + z, -y^2 + xz, -x^2 + tz, t^2z - xy]
-------------------------------
  ReducedGBasis(I);
[t^3 - x, tx - y, ty - z, y^2 - xz, x^2 - tz, t^2z - xy]
-------------------------------
\\=========================  o=o=o=o  =========================//
```

#8 - 25 May 2023 11:41 - Anna Maria Bigatti

John Abbott wrote:

> A possible advantage of using "funny" indet names is that it is then clear that the value does not belong to the original poly ring.
> A disadvantage is that debugging could be a real headache...

That's why I'm undecided. I'll follow my approach (when/after finished) I'll try them both and see how I feel about using them. (this should be a small change in the code)

> One the phone I had mentioned the possibility of "exporting" an internal polynomial to the original ring. This would automatically use the expected indet names, **but** the leading PP may not be the first one printed (if the internal ring has a different term order).

I think we should output the internal representation, and also offer the homomorphism to bring it in the input ring.
So I can write phi(LPP), or LPP), depending on what I need to look at.

> An "export" function would be useful/needed when producing a truncated GB. Also I have an application (find extra gens prior to computing the homogenization) where I want to find quickly some "small" polys in the ideal *e.g.* compute a "partial" GB with a time limit, then check to see if there are any small polys. Again this would require exporting the poly... it is no importance that the LPP may not be the first.

Ideally this framework should have no overhead compared to the usual GBasis call, so that we can implement all the GB variations using it (truncation, MinGens, ...)

#9 - 25 May 2023 14:18 - Anna Maria Bigatti

John Abbott wrote:

> A possible advantage of using "funny" indet names is that it is then clear that the value does not belong to the original poly ring.
> A disadvantage is that debugging could be a real headache...

That's why I'm undecided. I'll follow my approach (when/after finished) I'll try them both and see how I feel about using them. (this should be a small change in the code)

> One the phone I had mentioned the possibility of "exporting" an internal polynomial to the original ring. This would automatically use the expected indet names, **but** the leading PP may not be the first one printed (if the internal ring has a different term order).

I think we should output the internal representation, and also offer the homomorphism to bring it in the input ring.
So I can write phi(LPP), or LPP), depending on what I need to look at.

> An "export" function would be useful/needed when producing a truncated GB. Also I have an application (find extra gens prior to computing the homogenization) where I want to find quickly some "small" polys in the ideal *e.g.* compute a "partial" GB with a time limit, then check to see if there are any small polys. Again this would require exporting the poly... it is no importance that the LPP may not be the first.

Ideally this framework should have no overhead compared to the usual GBasis call, so that we can implement all the GB variations using it (truncation, MinGens, ...)

#10 - 25 May 2023 14:30 - Anna Maria Bigatti

Currently in CoCoA all GB-related computations are done in this way:

1. prepare an ideal/ideal-like submodule in a specific ring/FreeModule (with ordering)
2. compute the GBasis of this ideal/submodule
3. extract the information from it and convert it into the desired result.

Thus the GroebnerMill should have all the information (homomorphisms, actual operations, meaning of indeterminates, ...) to extract and convert the data from within the GBasis computation in the core.