Tutorial 2. Basic Algorithms.

June 7th, 2011

The goal of this tutorial is to implement an algorithm for involutive completion. Several algorithms can be implemented and several improvements can be made to the basic algorithms. It is also interesting to analyze the algorithms for different divisions, and see how the properties of the different divisions can be exploited to optimize and adapt the main algorithms.

Exercise 1:

Implement an algorithm for involutive completion. Depending on your interests you can choose at least one of the following alternatives:

- Implement the basic algorithm for monomial completion.
- Implement the basic algorithm for polynomial completion.
- Implement the $\mathcal{T} - \mathcal{Q}$ algorithm.

The $\mathcal{T} - \mathcal{Q}$ algorithm is the best choice in terms of efficiency and is not more complicated to implement than the basic algorithms. An interesting exercise is to compare the performance of different algorithms for some inputs.

Exercise 2: Analysis of the implemented algorithm(s). Any choice in exercise 1 needs to implement some subprograms and to perform intermediate computations, and depending on the division used and the actual set on which the algorithm is run, these intermediate computations grow to a big or small extent.

In this exercise one can concentrate on the auxiliar subprograms and observe the effect of optimizations on them.

To check this one can use a set of examples and note down running times, or use a C++ profiler (if you are implementing in CoCoALib) to see how many times subprograms are called, how much time they take, etc.. Also, one can set counters in the program (both in CoCoA and CoCoALib) to investigate function calls, size of interemediate computations, etc.

Another way of optimizing the algorithms is to make specific versions for some particualr divisions. Investigate where is it worth to make specific cubprograms if the division is global, or have separate subprograms for computing multiplicative indices, etc...