A Certified Version of Buchberger's Algorithm

Laurent Théry

INRIA, 2004 route des Lucioles, 06902 Sophia Antipolis France
thery@sophia.inria.fr

Abstract. We present a proof of Buchberger's algorithm that has been developed in the Coq proof assistant. The formulation of the algorithm in Coq can then be efficiently compiled and used to do computation.

1 Introduction

If we look at the way one can use computers to do mathematics, there is a clear separation between computing, where one uses computer algebra systems, and proving, where one uses theorem provers. The fact that these two aspects are covered separately has obvious drawbacks. On the one hand it is a well-known fact that, because of misuse or implementation errors, one should always double-check the results given by a computer algebra system. This problem is even more crucial for general-purpose computer algebra systems where the library of algorithms is mostly developed by the user community. Extensions of the system are then performed without giving evidence of their applicability or correctness. On the other hand theorem provers usually come with very little computing power. This makes it difficult to complete proofs for which some computing steps are needed.

It would be a real progress if one could unify these two aspects in a single system. It would then be possible to define mathematical objects and both compute and prove properties about them. Building such a system from scratch requires an important effort. A more pragmatic approach consists in complementing existing systems. If we look at computer algebra systems, the situation is somewhat difficult. The languages of general-purpose computer algebra systems have not been designed with the idea that people would like to reason about them. For example, the scope of local variables in Maple [2] is not limited to the procedure where they have been defined. Thus, stating properties of algorithms turns out to be very difficult.

If we look at theorem provers, the main problem is efficiency. While most theorem provers allow us to define algorithms, executing them is inefficient because it is performed inside the prover in an interpretative way. An alternative solution is for the prover to be able to translate its algorithms into another programming language that has a compiler.

Our approach follows the second line. We have chosen the theorem prover Coq [12] to do our experiments. Coq is a prover based on type theory. It manipulates
objects with a rich notion of types which is clearly adequate for mathematical objects. Coq also proposes an extraction mechanism that, given an algorithm defined in the system, generates an implementation in the language Ocaml [14] that can be efficiently compiled.

Is this solution practical? What is the effort involved in trying to certify standard algorithms for computer algebra systems? It is to answer these questions that we decided to work on the proof of correctness of Buchberger's algorithm. We started from a five page description of the algorithm in a standard introduction book [7]. The goal was simple: to develop enough mathematical knowledge in Coq for stating the algorithm and proving its correctness and termination.

The paper is organized as follows. In Section 2 we introduce the Buchberger's algorithm. In Sections 3 and 4 we sketch its proofs of correctness and termination. In Section 5 we explain the main steps of our development and give a running example of the algorithm. Finally we relate our approach to others and draw some conclusions and future work.

2 Buchberger's Algorithm

Buchberger's algorithm is a completion algorithm working on polynomials. Given a list of polynomials it returns a completed list that has a particular property. Before presenting the algorithm, we first need to define some basic notions [7].

2.1 Ordered Polynomials

We first consider the usual \( n \) variables polynomials over an arbitrary field \((A, +a, -a, *a, /a, 0a, 1a)\) with two of their usual operations: addition (+) and multiplication by a term (\( . \)). A polynomial is composed of a list of terms. Each term is composed of a coefficient and a monomial. The set of coefficients is \( A \). The set of monomials is denoted by \( M_n \) where \( n \) is the number of variables. The set of terms and polynomials are denoted by \( T_{A, M_n} \) and \( P_{A, M_n} \) respectively.

An order \( \leq_{M_n} \) over monomials is a binary relation that is transitive, reflexive and antisymmetric. It is total if two distinct elements are always comparable. It is well-founded if there exists no infinite strictly decreasing sequence of monomials. Finally it is admissible if \( x_1 \ldots x_n \) is minimal for the order and if the order is compatible with the multiplication.

Given an admissible well-founded total order \( \leq_{M_n} \) over monomials, it is possible to represent a polynomial as a list of terms, such that the list of the corresponding monomials is ordered, i.e. each monomial in the list is strictly greater than the ones at its right. We use \( 0 \) and \( + \) to denote the null polynomial and the ordered list constructor respectively. From this representation we get the structural induction theorem for an arbitrary predicate \( P \) over polynomials:

\[
(P 0) \land (\forall a \in A, \forall p \in P_{A, M_n}, (P p) \Rightarrow (P (a + p))) \Rightarrow \forall p \in P_{A, M_n}, (P p)
\]

We define the transitive relation \( <_p \) over polynomials as the smallest relation such that: