Gröbner Basis Construction Algorithms Based on Theorem Proving Saturation Loops

Grant Olney Passmore[1], Leonardo de Moura[2] and Paul B. Jackson[1]

[1] LFCS, University of Edinburgh
[2] Microsoft Research, Redmond

Abstract. We present novel Gröbner basis algorithms based on saturation loops used by modern superposition theorem provers. We illustrate the practical value of the algorithms through an experimental implementation within the Z3 SMT solver.

1 Introduction

Gröbner bases are an indispensable cornerstone of modern algorithmic algebra. Though originating as a tool for solving difficult algebro-geometric problems, the past twenty years have seen tremendous growth in the applications of Gröbner bases to areas as diverse as loop invariant generation [21], integer programming [23], reachability in Petri nets [8], and as crucial components to a number of recent decision methods for nonlinear real arithmetic [18, 19, 24]. In the work that follows, we are principally motivated by the use of Gröbner bases in the context of a breed of automated theorem provers known as *satisfiability modulo theories* (SMT) solvers. In particular, this study began with our wish to use Gröbner basis calculations within the high-performance SMT solver Z3 [16].

In attempting to integrate Gröbner basis calculations within Z3, it was observed by the authors that the known Gröbner basis procedures used for solving difficult algebro-geometric problems and available in modern computer algebra systems, such as Buchberger's algorithm [4] and its enhancements F4 and F5 [12, 13], were not able to cope with the flavour of large systems of polynomial constraints generated by the SMT solver. These types of nonlinear systems, usually derived from industrial software verification conditions, often contain massive ($> 1,000$) numbers of polynomial equations, but have a proportionally small nonlinear component. We call these types of systems 'large, largely linear' or 'L3' nonlinear systems. This paper is focused on the development of novel Gröbner basis calculation algorithms which allow us to cope with these L3 systems.

Tasked with the problem of constructing new Gröbner basis calculation algorithms tailored to the needs of SMT solvers, a very pleasing solution presented itself: We were able to exploit years of work undertaken within the automated theorem proving community and adapt saturation loops and fast term indexing techniques used by modern superposition theorem provers to the context of Gröbner basis calculation.

Dagstuhl Seminar Proceedings 10161
Decision Procedures in Software, Hardware and Bioware
http://drops.dagstuhl.de/opus/volltexte/2010/2734

These loops, one derived from McCune's OTTER [15], the other from Avenhaus et al's DISCOUNT [1], combined with sophisticated term indexing, have enabled modern high-performance theorem provers to reason effectively in the context of massive clause sets [20]. By adapting these developments to a Gröbner basis setting, we are able to leverage work done in one community to aid another. Indeed, these new algorithms allow us to cope Gröbner bases for systems much larger than those amenable to previously available Gröbner basis algorithms, provided that these systems contain a relatively small nonlinear component. While mapping these saturation loops to a Gröbner basis setting is straightforward, both proving their correctness and deriving appropriate term indexing techniques is not. To prove correctness of the top-level algorithms and justify the term indexing techniques described, we make use of the theory of Abstract Gröbner Bases [17].

2 Related Work and Novelty of Contribution

There is a rich history of work on connections between Gröbner bases, critical-pair completion, and automated theorem proving. Already in 1983 [6], the view of Gröbner basis construction as critical-pair completion was recognised by Buchberger and used to fruitfully extend Gröbner basis methods to new domains. Following this, Buchberger made connections between these ideas and resolution theorem proving explicit in 1987 [7]. The work most relevant to this paper, however, concerns *abstract frameworks* for analysing both completion and Gröbner basis procedures. In the case of completion, for instance, such a framework allows one to view different completion algorithms as being particular *strategies* for sequencing a small set of inference rules. In doing so, one is able use uniform methods for proving results about a multitude of different completion procedures simultaneously. There are a number of frameworks upon which we build.

The first is the Bachmair-Dershowitz theory of *Abstract Completion* [2]. The second is the Bachmair-Ganzinger framework developed for presenting Buchberger's algorithm as a *constraint-based* completion procedure [3]. In the end, we found it necessary to derive our own framework, based upon [2], for analysing the Gröbner basis algorithms presented in this paper. We call this framework *Abstract Gröbner Bases*. The primary impediment to using [2] or [3] directly for our purposes was that neither allowed us to easily investigate the admissibility of *superfluous S-polynomial criteria* in a strategy-independent setting. These critera, as explained below, are crucial to the term indexing techniques used in the algorithms presented. A detailed account of Abstract Gröbner Bases and the superfluous S-polynomial criteria is beyond the scope of this paper and may be found in [17, 11].

Finally, the idea of using sophisticated simplification and term indexing techniques during Gröbner basis construction has been explored by many, though the latter usually under a different name: as Gröbner basis procedures deal solely with polynomials, the phrase "term indexing" is usually eschewed in Gröbner basis research in favor of "polynomial representation" [9]. For example, the (very

different) techniques underlying both Faugère's F4 [12] and the Cory-Rossin-Salvy "sandpiles" method [10] may be seen as combining sophisticated simplification and term indexing [9].

2.1 Main Contribution

Given that two core ideas explored in this work – using sophisticated simplification and term indexing techniques during Gröbner basis construction – have been explored by many, we find it prudent to make clear which aspects of this work are novel.

Our main contribution is the particular *instantiation* of these ideas. This instantiation has been motivated by the types of problems encountered when using the Z3 SMT solver to verify programs with nonlinear arithmetical components and is particularly interesting from the perspective of automated theorem proving. While a number of prior works have put forth theoretical frameworks for building specialised Gröbner basis procedures, we are aware of none which actually *apply them* and present the details of such a specialisation from algorithm description and correctness to implementation and empirical evaluation. By focusing on specific saturation loops which have been successful in automated theorem proving and mapping them to a Gröbner basis setting, and by undertaking this work in the context of a high-performance SMT solver, we provide a foundation upon which other SMT solver researchers may build. Similarly, we feel this work gives a tangible basis for researchers in automated theorem proving to consider how other techniques in their repertoire may be imported to a Gröbner basis setting.

3 Mathematical Preliminaries

3.1 Notation

Given $\{p_1, \ldots, p_k\} \subset \mathbb{Q}[\boldsymbol{x}]$, the polynomial ideal $\mathcal{I}(\{p_1, \ldots, p_k\})$ is the set of polynomials $\left\{\sum_{i=1}^{k} p_i q_i \mid q_i \in \mathbb{Q}[\boldsymbol{x}]\right\}$. An element $x_1^{i_1} \ldots x_n^{i_n}$ in $\mathbb{Q}[x_1, \ldots, x_n]$ is called a *power-product* (or *term*), and an element $c x_1^{i_1} \ldots x_n^{i_n}$ with $c \in \mathbb{Q}$ and $x_1^{i_1} \ldots x_n^{i_n}$ a power-product is called a *monomial*. We say a monomial is *monic* if $c = 1$. Observe that power-products may be seen as monic monomials. We use $\mathbb{M}$ to denote the set of all power-products in $\mathbb{Q}[x_1, \ldots, x_n]$. From hereafter, we use p, q and r to denote polynomials, m to denote monic monomials, c to denote coefficients, and cm to denote monomials. We assume all polynomials are in sum-of-monomials normal form (e.g., a polynomial will never contain two distinct monomials formed from the same power-product).

We say a power-product $x_1^{i_1} \ldots x_n^{i_n}$ *contains* x_k if $i_k > 0$. Given two power-products $m_1 = x_1^{i_1} \ldots x_n^{i_n}$ and $m_2 = x_1^{j_1} \ldots x_n^{j_n}$, $m_1 m_2$ denotes the power-product $x_1^{i_1+j_1} \ldots x_n^{i_n+j_n}$, if $i_k \geq j_k$ for all $k \in \{1, \ldots, n\}$, then $\frac{m_1}{m_2}$ denotes the power-product $x_1^{i_1-j_1} \ldots x_n^{i_n-j_n}$, and the *least common multiple* $\mathsf{lcm}(m_1, m_2)$ of m_1 and m_2 is the power product $x_1^{max(i_1,j_1)} \ldots x_n^{max(i_n,j_n)}$.

A total order $\prec$ on the set $\mathbb{M}$ is *admissible* if $m_1 \prec m_2$ implies that $m_1 m \prec m_2 m$, for all m_1, m_2 and m in $\mathbb{M}$. A *monomial order* is a total order on $\mathbb{M}$ which is admissible and a well ordering. Given two polynomials p_1 and p_2, we say $p_1 \prec p_2$ if there is a monomial cm contained in p_2 s.t. (i) m is not contained in p_1, and (ii) for all m' contained in p_1, if $m \prec m'$, then m' is contained in p_2. If $p = cm + q$ with $q \prec cm$, then we say cm is the *leading monomial* of p ($LM(p) = cm$) and we may write this as $p = \underline{cm} + q$.

Given a monomial order $\prec$ and two monic monomials $p_1 = \underline{m_1} + q_1$ and $p_2 = \underline{m_2} + q_2$, let $\tau_{1,2} = \mathsf{lcm}(m_1, m_2)$. Then, we use $\mathsf{spol}(p_1, p_2)$ to denote the polynomial $(\frac{\tau_{1,2}}{m_1})q_1 - (\frac{\tau_{1,2}}{m_2})q_2$. Given a set of polynomials S, it is easy to see that if $\{p_1, p_2\} \subseteq \mathcal{I}(S)$, then $\mathsf{spol}(p_1, p_2) \in \mathcal{I}(S)$.

3.2 Abstract Gröbner Bases

We now briefly present portions Abstract Gröbner Bases relevant to this work.

Why go abstract? Abstract Gröbner Bases was introduced as a framework for proving the correctness of Gröbner basis algorithms w.r.t. a multitude of execution strategies. But what is a strategy? Perhaps the best way to answer this question is to examine Buchberger's algorithm (Fig. 2) and consider *which aspects of it might be changed* while still preserving its correctness. Observe, for instance, that once a polynomial is placed in the growing Gröbner basis G, it is never removed. Notice as well that members of G are only used to simplify derived S-polynomials, and never the other way around.

In automated theorem proving, it has long been recognised that sophisticated simplification methods must be used to handle large sets of deduced clauses. For this reason, high-performance saturation loops keep two lists of clauses, *active* and *passive*, using *active* clauses for inferring new facts, and using a number of different simplification techniques (*forward* and *backward* simplification) for reducing the complexity of kept clauses. When working to solve L3 systems, it will be shown that such simplification methods can also be adapted to help compute large Gröbner bases.

A sketch of the theory Given a monomial order $\prec$, the key idea underlying Gröbner bases is to use a polynomial $\underline{cm} + q$ as a rewrite rule $cm \to -q$. To simplify the presentation that follows, we will assume all polynomials used as rewrite rules are monic. The monic polynomial $p = \underline{m} + q$ induces a *reduction relation* $\mapsto_p$ on polynomials. It is defined as $q_1 + c_1 m_1 m \mapsto_p q_1 - c_1 m_1 q$ for arbitrary monomials $c_1 m_1$ and polynomials q_1. Given a set of monic polynomials $G = \{p_1, \ldots, p_k\}$, the reduction relation induced by G is defined as: $\mapsto_G = \bigcup_{i=1}^{k} \mapsto_{p_i}$.

Definition 1 (Gröbner bases). *A finite set of monic polynomials G is a Gröbner basis of the ideal $\mathcal{I}(F)$ iff $\mathcal{I}(G) = \mathcal{I}(F)$ and $\mapsto_G$ is confluent.*

$$\textsf{Orient} \quad \frac{S \cup \{\underline{cm} + q\}, G}{S, G \cup \{\underline{m} + (\frac{1}{c})q\}}$$

$$\textsf{Superpose} \quad \frac{S, G \cup \{p_1, p_2\}}{S \cup \{\textsf{spol}(p_1, p_2)\}, G \cup \{p_1, p_2\}}$$

$$\textsf{Delete} \quad \frac{S \cup \{0\},\ G}{S, G}$$

$$\textsf{Simplify-S} \quad \frac{S \cup \{c_1 m_1 m_2 + q_1\}, G \cup \{\underline{m_2} + q_2\}}{S \cup \{q_1 - c_1 m_1 q_2\}, G \cup \{\underline{m_2} + q_2\}}$$

$$\textsf{Simplify-H} \quad \frac{S, G \cup \{\underline{m_1 m_2} + q_1,\ \underline{m_2} + q_2\}}{S \cup \{q_1 - m_1 q_2\}, G \cup \{\underline{m_2} + q_2\}} \ \textbf{if } m_1 \neq 1$$

$$\textsf{Simplify-T} \quad \frac{S, G \cup \{\underline{m} + c_1 m_1 m_2 + q_1,\ \underline{m_2} + q_2\}}{S, G \cup \{\underline{m} - c_1 m_1 q_2 + q_1,\ \underline{m_2} + q_2\}}$$

Fig. 1. Abstract GB calculus

The inference rules in Figure 1 work on pairs of sets of polynomials (S, G). In all rules, the coefficients c and c_1 are assumed to be non-zero. We use $(S_1, G_1) \vdash (S_2, G_2)$ to indicate that (S_1, G_1) can be transformed to (S_2, G_2) by applying one of the inference rules in Figure 1.

Theorem 1. *$(S_1, G_1) \vdash (S_2, G_2)$ implies $\mathcal{I}(S_1 \cup G_1) = \mathcal{I}(S_2 \cup G_2)$.*

Definition 2 (Procedure). *A Gröbner basis procedure $\mathfrak{G}$ is a program that accepts a set of polynomials $\{p_1, \ldots, p_k\}$, a monomial order $\prec$, and uses the rules in Figure 1 to generate a (finite or infinite) sequence $(S_1 = \{p_1, \ldots, p_k\},\ G_1 = \emptyset) \vdash (S_2, G_2) \vdash (S_3, G_3) \vdash \ldots$. This sequence is called a* run *of $\mathfrak{G}$.*

Given a set of monic polynomials G, the set of S-polynomials $\textsf{SP}(G)$ is defined as the set

$$\{\textsf{spol}(p_1, p_2) \mid p_1, p_2 \in G\}.$$

Definition 3 (Correct Procedure). *A Gröbner basis procedure $\mathfrak{G}$ is said to be correct iff it produces only finite runs $(S_1,\ G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset,\ G_n)$, and*

$$\textsf{SP}(G_n) \subseteq (S_1 \cup S_2 \cup \ldots \cup S_{n-1})\,.$$

Theorem 2. *Let $\mathfrak{G}$ be a correct Gröbner basis procedure, then for any run $(S_1,\ G_1 = \emptyset) \vdash \ldots \vdash (S_n = \emptyset,\ G_n)$, G_n is a Gröbner basis for $\mathcal{I}(S_1)$.*

Definition 4 (Eager S-simplification). *Given a Gröbner basis procedure $\mathfrak{G}$, we say $\mathfrak{G}$ implements eager S-simplification iff $\mathfrak{G}$ only applies* Orient *to $p \in S_i$ when* Simplify-S *cannot be applied to p.*

Definition 5 (Fairness). *A Gröbner basis procedure* $\mathfrak{G}$ *is said to be fair iff for any run* $(S_1, G_1) \vdash (S_2, G_2) \vdash \ldots$

$$\mathsf{SP}(\bigcup_{i\geq 1}\bigcap_{j\geq i} G_j) \subseteq \bigcup_{i\geq 1} S_i.$$

Theorem 3. *If a Gröbner basis procedure* $\mathfrak{G}$ *implements eager S-simplification, is fair, and* Superpose *is applied at most once for any pair of polynomials in* $\bigcup_{i\geq 1} G_i$*, then* $\mathfrak{G}$ *is correct.*

Observe that the rule Superpose may be adjusted to take into account side-conditions barring its application. Such side-conditions correspond to *superfluous S-polynomial criteria* in Gröbner basis theory and an account is given in [17]. The notion of *eager SH-simplification* will be used for term indexing.

Definition 6 (Eager SH-simplification). *We say a Gröbner basis procedure* $\mathfrak{G}$ *implements eager SH-simplification iff* $\mathfrak{G}$ *only applies* Orient *to* $p \in S_i$ *when* Simplify-S *cannot be applied to* p*, and* $\mathfrak{G}$ *only attempts*[3] *to apply* Superpose *to* $p_1, p_2 \in G_i$ *when* Simplify-H *cannot be applied to* p_1, p_2*.*

To gain familiarity with the rules, we present an Abstract GB implementation of Buchberger's algorithm in Fig. 3.

Input: $\langle F = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\boldsymbol{x}], \prec\rangle$
Output: G s.t. G is a GBasis of F w.r.t. $\prec$
G := F; S := $\{\langle p_i, p_j\rangle \mid 1 \leq i < j \leq k\}$
while $S \neq \emptyset$ **do**
  Let $\langle p_i, p_j\rangle \in S$
  For some q s.t. *S-polynomial*$(p_i, p_j) \xrightarrow{G} q$
  if $q \neq 0$ **then**
    $S := S \cup \{\langle p, q\rangle \mid p \in G\}$
    $G := G \cup \{q\}$
  end if
  $S := S \setminus \{\langle p_i, p_j\rangle\}$
end while

Fig. 2. Buchberger's Algorithm

4 Algorithms: OTTER-GB and DISCOUNT-GB

We now describe two new algorithms for computing Gröbner bases. These algorithms are aimed at solving L3 systems which are beyond the reach of previously available methods. There are two main procedural ingredients to these

[3] By "attempts to apply" we mean that Superpose is either applied as usual, or it is tried but is ultimately skipped because of an active side-condition φ barring its application.

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\boldsymbol{x}], \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
Apply Orient to every member of S
Apply Superpose between every $p_i, p_j \in G$ $(p_i \neq p_j)$
while $S \neq \emptyset$ **do**
  Choose spol$(p_i, p_j) \in S$
  Apply Simplify-S to spol$(p_i, p_j) \in S$ as long as possible
  Call the resulting simplified polynomial (in S) q
  if $q \neq 0$ **then**
    Apply Orient to q
    Apply Superpose to all pairs $\langle p, q \rangle$ $(p \neq q \in G)$
      for which Superpose has not been previously
      applied
  else
    Apply Delete to q
  end if
end while

Fig. 3. Rule-based Simulation of Buchberger's Algorithm

algorithms: (i) top-level loops adapted from the OTTER and DISCOUNT saturation algorithms, and (ii) term indexing techniques derived from both the theorem proving literature [22] and so-called 'superfluous S-polynomial criteria' which are important in Gröbner basis theory. The term indexing techniques are designed for facilitating fast applications of the inference rules Superpose, Simplify-S, Simplify-T, and Simplify-H. It will be easily seen that these algorithms correspond formally to *correct strategies* in the sense of of Abstract GBs. Thus, by Theorem 2, they are guaranteed to be terminating, functionally correct Gröbner basis construction algorithms.

4.1 Understanding the algorithms

To help understand these algorithms, it is instructive to examine some differences between them and both Buchberger's algorithm and Faugére's F4. But before doing so, let us reflect on how OTTER-GB and DISCOUNT-GB differ from each other. Through every iteration of OTTER-GB, both G and S are kept maximally simplified w.r.t. G. Contrast this with DISCOUNT-GB in which only G is kept maximally simplified w.r.t. G. If we associate G with `active` and S with `passive`, this is precisely the fundamental difference between the OTTER and DISCOUNT saturation loops. As in Figs. 4 and 5, use `new` to denote the result of applying an inference procedure to deduce new facts (perhaps *resolution* in the case of theorem proving and the *computation of S-polynomials* in the case of Gröbner bases). In saturation vernacular, using members of `active` to simplify members of `new` in the OTTER loop, as well as the picked polynomial p in the DISCOUNT loop, is called *forward simplification*. Using the current/picked polynomial q to simplify members of `active` $\cup$ `passive` is called *backward sim-*

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\boldsymbol{x}], \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
while $S \neq \emptyset$ **do**
  Invariant: G and S are maximally simplified w.r.t. G
  Choose $p \in S$
  Apply Orient to p
  Let q be the resulting oriented polynomial (in G)
  Use q to simplify G as long as possible
    using Simplify-H and Simplify-T
  Use G to simplify S as long as possible
    using Simplify-S
  Let $\mathtt{S_{old}} := S$
  Apply Superpose to all pairs $\langle p_i, q \rangle$ $(p_i \neq q \in G)$
    for which Superpose has not been previously
    applied
  Let $\mathtt{new} := S \setminus \mathtt{S_{old}}$
  Use G to simplify members of S in `new`
    as long as possible using Simplify-S
  Apply Delete if possible
  if $((G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset)$ **then**
    return $\{1\}$
  end if
end while
`return` G

Fig. 4. GB algorithm based on OTTER saturation loop

Input: $\langle S = \{p_1, \ldots, p_k\} \subset \mathbb{Q}[\boldsymbol{x}], \prec \rangle$
Output: G s.t. G is a GBasis of S w.r.t. $\prec$
while $S \neq \emptyset$ **do**
  Invariant: G is maximally simplified w.r.t. G
  Choose $p \in S$
  Use G to simplify p as long as possible
    using Simplify-S
  Let s be the resulting simplified polynomial (in S)
  if $s \neq 0$ **then**
    Apply Orient to s
    Let q be the resulting oriented polynomial (in G)
    Use q to simplify G as long as possible
      using Simplify-H and Simplify-T
    Apply Superpose to all pairs $\langle p_i, q \rangle$ $(p_i \neq q \in G)$
      for which Superpose has not been previously
      applied
    Apply Delete if possible
    if $((G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset)$ **then**
      return $\{1\}$
    end if
  else
    Apply Delete to s
  end if
end while
`return` G

Fig. 5. GB algorithm based on DISCOUNT saturation loop

plification. We will see that in this sense, Buchberger's algorithm performs only *forward simplification*, whereas OTTER-GB and DISCOUNT-GB perform both.

In our working analogy between Gröbner basis construction and saturation theorem proving, we have the following intuitive correspondence.

ATP	**GB**
unit clause	polynomial
`infer`	Superpose
resolvent	S-polynomial
`active`	G
`passive`	S
forward simp.	Simplify-S
backward simp.	Simplify-H, Simplify-T
subsumed resolvent	superfluous S-polynomial

Thus, we see fundamentally how Buchberger's algorithm differs from those presented here. In Buchberger's algorithm, once a reduced S-polynomial is placed in G, it is never removed. It only further participates in the algorithm by being used as a simplifier to reduce newly computed S-polynomials. Thus, Buchberger's algorithm only performs forward simplification. Both OTTER-GB and DISCOUNT-GB perform forward and backward simplification, but do so in different ways. In OTTER-GB, backward simplification is applied to both G and S, whereas in DISCOUNT-GB it is applied only to G.

While also only performing forward simplifications, Faugére's F4 differs from Buchberger's algorithm in that it (using deep insights from linear algebra) simplifies many S-polynomials simultaneously. In a primitive sense, a related phenomenon happens in the algorithms we present. Once a chosen polynomial is used to compute a set of S-polynomials against the other members of G, any of the deduced S-polynomials may become an immediate target for simplification, and the simplification steps of multiple S-polynomials may be interleaved.

4.2 Term indexing

As with terms in high-performance automated theorem proving [22], it is imperative to have efficient methods for computing sets of polynomials which match a given polynomial w.r.t. the Abstract GB inference rules. These techniques need to answer queries of the form "which polynomials in X can be used to perform an inference using rule R with polynomial p?".

It is fair to say that without such indexing methods, our new procedures would likely not perform better than those which were previously available. Both the adapted saturation loops and the term indexing are crucial to the improved performance.

Indexing for Superpose For the application of Superpose, the indexing technique is based on superfluous S-polynomial criteria in Gröbner basis theory. Such

a criterion is a computationally efficient sufficient condition for recognising when a given S-polynomial would reduce to zero w.r.t. the Gröbner basis being constructed, and thus can be ignored. Such a criteria is in a sense a subsumption check, as an S-polynomial reducing to zero implies that all reductions it induces are present in the rewrite system induced by the portion of the Gröbner basis already constructed. Hence it would not contribute to obtaining a confluent rewriting system and need not be considered.

There is, however, a difficulty in using such criteria in non-standard Gröbner basis loops: classical criteria, such as Buchberger 1 and 2, were originally proved correct only w.r.t. a fixed basis construction strategy, e.g., using an inductive cut-point argument w.r.t. the classical Buchberger's algorithm [5]. This problem has recently been addressed in [17] where both of these criteria are proved to be admissible in the context of Abstract GBs.

In our implementation, we make use of Criteria[4] 1 and 3 in [17]. Letting $p_1 = \underline{m_1} + q_1, p_2 = \underline{m_2} + q_2$:

Criteria 1 (Buchberger-1) *If* $lcm(m_1, m_2) = m_1 m_2$ *then* $\mathsf{spol}(p_1, p_2)$ *is superfluous.*

Criteria 3 *If* p_1, p_2 *have been eagerly SH-simplified and* $m_1 | m_2$ *or* $m_2 | m_1$*, then* $\mathsf{spol}(p_1, p_2)$ *is superfluous.*

It follows from Criteria 3 that polynomials with linear leading monomials need not participate in Superpose inferences, given that both algorithms presented implement eager SH-simplification. Thus, for the application of Superpose, we index polynomials by their leading monomials so that given a polynomial p, we may quickly return lists of other polynomials whose leading monomials are (i) not relatively prime to p, and (ii) nonlinear. The index is essentially tracking the occurrences of variables in leading monomials of polynomials in G which have the potential to mate with p to contribute non-superfluous S-polynomials.

Indexing for forward simplification For the application of forward simplification (Simplify-S), the indexing technique is based on the following observation.

Observation 1 *For an oriented polynomial* $p_2 = \underline{m_2} + q_2 \in G$ *to be used to simplify an unoriented* $p_1 = c_1 m_1 m_2 + q \in S$ *using* Simplify-S*, it follows that (i)* $\mathsf{totaldeg}(m_2) \leq \mathsf{totaldeg}(m_1 m_2)$*, and (ii) every variable in* m_2 *must appear in* $m_1 m_2$*.*

While the above observation may seem a triviality, in practice it implies that to find an oriented polynomial which may Simplify-S a target monomial, we may place oriented polynomials in an index which facilitates (i) only considering polynomials whose leading monomial's total degree does not surpass that of the

[4] In fact, the statement of Criteria 3 in this work is really the combination of Criteria 3 and Theorem 9 in [17] together with the fact that polynomials with linear leading monomials in different variables need not be Superposed (corollary of Criteria 1).

target, and (ii) in doing so only one variable of each leading monomial need be indexed. To build such an index `fw_index`: `Nat` $\times$ `Nat` $\to 2^{\mathbb{Q}[\boldsymbol{x}]}$ we process each $p \in G$ as follows (where `#`: `Var` $\to$ `Nat` is an appropriate injection):

> Let `v` be some variable in the leading monomial of p
> Let `n` be the power of `v` in this monomial in p
> Add p to `fw_index[#(v)][n]`

Now, to find a polynomial p which can rewrite a monomial m, we may perform a restricted search:

for each variable `v` in m **do**
  Let `deg` = totaldeg(m)
  for `n` in $[1 \ldots$ `deg`$]$ **do**
    for each `q` $\in$ `fw_index[#(v)][n]` **do**
      if $(LM(q)|m)$ **then**
        return q
      end if
    end for
  end for
end for

In practice for L3 systems, most polynomials have low degree. So, we add a threshold in our index, and don't distinguish in the index between exponents with values greater than the threshold. In our implementation, variables are encoded as natural numbers, so the injection `#`: `Var` $\to$ `Nat` is just the identity function.

Indexing for backward simplification The index used for backwards simplification (Simplify-H and Simplify-T) is the most expensive of the three, as it requires we index every monomial. This index is slightly different depending on the loops, as in DISCOUNT-GB we only need to backwards simplify G, and so less targets for backward simplification need be indexed. The index `bw_index`: `Nat` $\times$ `Nat` $\to 2^{\langle \mathbb{Q}[\boldsymbol{x}],\texttt{Nat},\texttt{Nat}\rangle}$ maps a variable x and a degree d to a set of tuples of the form $\langle p, i, j\rangle$, where each tuple stores the fact that x is the j-th variable in the i-th monomial of p. Differently from `fw_index`, every variable in every monomial of p is indexed instead of just some variable in the lead monomial. Given $p = \underline{m} + q$, we find targets for simplification using the following algorithm.

```
v' = undef
min = UINT_MAX
n = totaldeg(m)
for each variable v in m do
  if (v' = undef  or  num_occs(v,n) < min) then
    v' = v
```

```
      min = num_occs(v,n)
    end if
  end for
```

At this point, `v'` is the variable with the least number of occurrences in monomials with degree at most `n`, and we assume `max` is the maximum multivariate total degree of all monomials which are possible targets (in L3 systems, `max` is usually low). We now finish finding targets as simplify.

```
for i in [n...max] do
  for each m' in bw_index[#(v')][i] do
    if (m|m') then
      Apply backward simplification
    end if
  end for
end for
```

In our implementation, the sets used in `fw_index` and `bw_index` are implemented using dynamic arrays (`vectors`). To efficiently remove elements from these indices, we allow empty entries in these dynamic arrays, so entries do not need to be moved when removed. We also store the value k in every polynomial p, when p is the k-th entry in the dynamic array $\texttt{fw\_index}[\#(v)][d]$. Moreover, we store the value k in the j-th variable of the i-th monomial of the polynomial p, when $\langle p, i, j\rangle$ is the k-th entry in the dynamic array $\texttt{bw\_index}[\#(v)][d]$. These additional data-structures allows us to update the indices in constant time. Note that, our indices are simpler than the ones used in saturation provers (e.g., Substitution Tree [14]) because our terms are shallow and ground.

4.3 Algorithm correctness

Theorem 4. *OTTER-GB and DISCOUNT-GB are terminating, functionally correct GB algorithms.*

Proof. By admissibility of Criteria 1 and 3, term indexing may be ignored. By the definition of polynomial ideal, if $(G \cup S) \cap (\mathbb{Q} \setminus \{0\}) \neq \emptyset$ *then* $\mathcal{I}(G \cup S) = \mathbb{Q}[\boldsymbol{x}]$. *Hence* $\{1\}$ *is a Gröbner basis for* $\mathcal{I}(G \cup S)$ *w.r.t. any* $\prec$ *and the* `return` $\{1\}$ *statement does not affect functional correctness. By Theorem 3, correctness is guaranteed if the procedures are* fair*, implement eager S-simplification, and Superpose is applied at most once between any two polynomials. The latter two properties are obvious. To observe fairness, we show* $\mathsf{SP}(\bigcup_{i\geq 1}\bigcap_{j\geq i} G_j) \subseteq \bigcup_{i\geq 1} S_i$. *Suppose not. Then, there must be some persistent pair* $p_1, p_2 \in \bigcup_{i\geq 1}\bigcap_{j\geq i} G_j$ *s.t.* $\mathsf{spol}(p_1, p_2) \notin \bigcup_{i\geq 1} S_i$. *WLOG, assume* $p_1 \in \bigcap_{j\geq k_1} G_j$ *and* $p_2 \in \bigcap_{j\geq k_2} G_j$ *s.t.* $(k_2 > k_1)$ *and let* k_1, k_2 *be the least indices with this property. Then* Simplify-H *and* Simplify-T *must not have been used to simplify* p_1, p_2 *beyond states* k_1, k_2 *resp., as this would violate persistence. Consider the pass of either loop in which the* Orient *step corresponding to state* k_2 *occurs. Since* p_1 *is also persistent in* G *beyond state* k_2*, it follows that in such a pass* Superpose *must have been applied*

between p_1 and p_2, or Superpose *must have been skipped because it had been previously applied to p_1, p_2. In either case we have* $\mathsf{spol}(p_1, p_2) \in \bigcup_{i \geq 1} S_i$. $\Longrightarrow\Longleftarrow$ □

5 Experimental results

To evaluate our implementation[5], we created sets of random benchmarks with 4 kinds of polynomials: (a) identity polynomials of form $x - y$, (b) difference polynomials of form $x - y + k$ where k is an integer constant, (c) general linear polynomials, and (d) general polynomials. We experimented with two distributions of these kinds, a *mostly-lin* distribution with 40%, 50%, 5%, 5% of the four kinds, which reflects distributions of L3 problems we see in practice, and a *non-lin* distribution with 100% of kind (d).

			discount		otter		m-fgb		m-f4		m-buchb	
Distrib	#Polys	#Vars	#i	avtm	#i	avtm	#i	avtm	#i	avtm	#i	avtm
mostly-lin	100	100	10	.00(10)	10	.00(10)	10	.05(10)	10	.14(10)	10	.45(10)
	100	200	3	.00(10)	3	.00(10)	3	.12(10)	3	.53(10)	3	1.30(10)
	100	500	1	.00(10)	1	.00(10)	1	.14(10)	1	.25(10)	1	1.27(10)
	1000	1000	10	.00(10)	10	.00(10)	0	TO	9	4.39(9)	0	6.96(7)
	1000	2000	4	.00(4)	4	.00(4)	0	TO	3	6.65(3)	0	TO
	1000	5000	0	.01(10)	0	.01(10)	0	TO	0	TO	0	TO
	10000	10000	10	.06(10)	10	.07(10)	0	TO	0	TO	0	TO
	10000	20000	10	.07(10)	10	.09(10)	0	TO	0	TO	0	TO
	10000	50000	4	.11(10)	4	.13(10)	0	TO	0	TO	0	TO
non-lin	10	5	8	.71(8)	9	.46(9)	10	.01(10)	10	.05(10)	10	.05(10)
	10	10	0	8.06(1)	0	2.08(2)	0	1.90(6)	0	1.74(4)	0	1.93(3)
	50	20	9	.61(9)	9	.34(9)	10	.05(10)	9	.24(9)	10	1.31(10)
	50	50	2	.00(2)	2	.00(2)	1	.03(1)	2	.18(2)	2	.27(2)
hard-a-g	5-8	5-8	0	TO	0	TO	0	.25(4)	0	3.91(3)	0	2.72(2)

Table 1. Experimental Results

Our results are summarised in Table 1. Each row but the last shows the results for 10 benchmarks with #Polys polynomials in #Vars variables. The last row shows results with 4 hard algebro-geometric benchmark problems which have been used to demonstrate the value of the F4 algorithm. The *discount* and *otter* columns are for our two implementations, and the other three are for Gröbner basis algorithms available in Maple 13: the *m-fgb* column is for a compiled implementation of the F4 algorithm written by J.C. Faugere, the *m-f4* column is for a Maple re-implementation of F4, and the *m-buchb* column for the traditional Maple implementation of the Buchberger algorithm. Each of the

[5] A tarball containing all of our experimental data, including implementation binaries, may be found at: http://research.microsoft.com/en-us/um/people/leonardo/dagstuhl-2010-experiments.tar.gz

entries in a column has 3 components: the number in the $\#i$ sub-column is the number of problems where the computed Gröbner basis is $\{1\}$, i.e. the set of polynomial equations is inconsistent, the 2nd number in the *avtm* sub-column is the average run-time in seconds for those problems on which tests halted in under 10 seconds, and the number in parentheses is the number of tests which halted in under 10 seconds. If the runs of all problems were over 10 seconds, the 2nd and 3rd numbers are replaced by TO for *time-out*.

As expected, as the ratio of polynomials to variables increases, we get more constrained systems and more definitely inconsistent problems. With the mostly-linear problems, as problem size increases, we see our procedures perform significantly better than those in Maple. One reason for this is that algorithms based on the principles of Buchberger's algorithm, which includes F4, have a quadratic preprocessing step of computing all initial pairs of non-identical polynomials of the input basis, whereas our algorithms do not require this for correctness.

We see with the general non-linear random problems the Maple algorithms are usually better, and, with the hard algebro-geometric problems, the Maple algorithms are far superior.

6 Future work

We see one immediate way in which these algorithms may be improved. This involves the mapping of Criteria 2 in [17] into an appropriate modification of our term indexing routines for Superpose. While this would likely not drastically affect performance on L3 systems, it seems plausible that this would be a boon in practice for computing with large systems of polynomials which have a higher nonlinear component than those currently amenable to our methods.

7 Conclusion

We have leveraged work within the automated theorem proving community to aid the extension of core computer algebra techniques to a challenging new type of problem. In particular, we have designed, implemented, and evaluated new Gröbner basis construction algorithms based on a combination of the OTTER and DISCOUNT saturation loops and term indexing techniques derived from both high-performance theorem proving and superfluous S-polynomial criteria in Gröbner basis theory. These procedures have been observed to significantly outperform previously available Gröbner basis algorithms for large, largely linear (L3) nonlinear systems. While proving these new algorithms correct was non-trivial, it became easy given the theory of Abstract Gröbner Bases. We see this work as an exciting cross-pollenation between core techniques of the theorem proving and computer algebra communities, and look forward to furthering this natural symbiosis.

References

1. Jürgen Avenhaus, Jörg Denzinger, and Matthias Fuchs. Discount: A system for distributed equational deduction. In *RTA '95: Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, pages 397–402, London, UK, 1995. Springer-Verlag.
2. L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *Journal of ACM*, 42(2), 1994.
3. Leo Bachmair and Harald Ganzinger. Buchberger's algorithm: A constraint-based completion procedure. In Jean-Pierre Jouannaud, editor, *CCL*, volume 845 of *Lecture Notes in Computer Science*, pages 285–301. Springer, 1994.
4. B. Buchberger. Ein algorithmus zum auffinden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal. Technical report, Mathematical Institute, University of Innsbruck, Austria, 1965.
5. Bruno Buchberger. A criterion for detecting unnecessary reductions in the construction of groebner bases. In *EUROSAM '79: Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, pages 3–21, London, UK, 1979. Springer-Verlag.
6. Bruno Buchberger. A critical-pair/completion algorithm for finitely generated ideals in rings. In *Proceedings of the Symposium "Rekursive Kombinatorik" on Logic and Machines: Decision Problems and Complexity*, pages 137–161, London, UK, 1984. Springer-Verlag.
7. Bruno Buchberger. History and basic features of the critical-pair/completion procedure. *J. Symb. Comput.*, 3(1-2):3–38, 1987.
8. O. Caprotti, A. Ferscha, and H. Hong. *Reachability Test in Petri Nets by Gröbner Bases.* Johannes Kepler University Linz Technical Report, 1995.
9. Jacques Carette. Personal communication, 2010.
10. Robert Cori, Dominique Rossin, and Bruno Salvy. Polynomial ideals for sandpiles and their gröbner bases. *Theor. Comput. Sci.*, 276(1-2):1–15, 2002.
11. L. de Moura and G. O. Passmore. On locally minimal nullstellensatz proofs. In *SMT '09: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, pages 35–42, New York, NY, USA, 2009. ACM.
12. Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases (f4). *Journal of Pure and Applied Algebra*, 1999.
13. Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (f5). In *ISSAC '02: Proceedings of the 2002 international symposium on Symbolic and algebraic computation*, pages 75–83, New York, NY, USA, 2002. ACM.
14. Peter Graf. Substitution tree indexing. In *RTA '95: Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, pages 117–131, London, UK, 1995. Springer-Verlag.
15. William McCune. Otter 2.0. In Mark E. Stickel, editor, *CADE*, volume 449 of *Lecture Notes in Computer Science*, pages 663–664. Springer, 1990.
16. Leonardo De Moura and Nikolaj Bjrner. Z3: An efficient SMT solver. In *TACAS'08*, 2008.
17. G. O. Passmore and L. de Moura. Superfluous S-polynomials in Strategy-Independent Gröbner Bases. In *SYNASC'09*, 2009.
18. G. O. Passmore and P. B. Jackson. Combined decision techniques for the existential theory of the reals. In *Calculemus'09: 16th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning*, 2009.

19. A. Platzer, J. Quesel, and P. Rümmer. Real world verification. In *CADE-22*, 2009.
20. Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *J. Symb. Comput.*, 36(1-2):101–115, 2003.
21. Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *ISSAC '04: Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, pages 266–273, New York, NY, USA, 2004. ACM.
22. R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. Term indexing. pages 1853–1964, 2001.
23. R. Thomas. Gröbner bases in integer programming. In D.-Z. Du and P.M. Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, 1998.
24. A. Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *CSL'05*, volume 3634 of *LNCS*, 2005.