SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Boolean polynomials and Gröbner bases: An algebraic approach to solving the SAT-problem

av

John Sass

2011 - No 4

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 106 91 STOCKHOLM

Boolean polynomials and Gröbner bases: An algebraic approach to solving the SAT-problem

John Sass

Självständigt arbete i matematik 30 högskolepoäng, Avancerad nivå

Handledare: Samuel Lundqvist

2011

Abstract

Being NP-complete, the problem of deciding boolean satisfiability, or SAT-problem, is probably impossible to solve efficiently. Nevertheless, this hasn't stopped people trying, and in this paper we present one method based on abstract algebra. The paper contains a repetition of basic abstract algebra, as well as an introduction to Gröbner bases and the Buchberger algorithm. Then we will show how boolean formulas can be transformed into polynomials in the $\mathbb{Z}_2$-polynomial ring, so that satisfiability is preserved as solvability. After that we will take a closer look at the ideal generated by such polynomials and the so called field equations, and prove that any potential solutions to this ideal only has coordinates in $\mathbb{Z}_2$, precisely corresponding to solutions that satisfy the boolean formula. After that, we will show how a quick glance at a Gröbner basis of this ideal will tell if any solutions exist. Finally, there will be some technical discussion, heuristic suggestions and possible ways to develop the technique further.

Contents

1 Introduction

The problem of boolean satisfiability, the SAT-problem, is probably the most famous and most discussed NP-complete problem. It looks quite simple on paper: given a boolean formula, such as $A \wedge (B \vee \neg A)$, is there any way to assign values to the variables so that the formula evaluates to true? In our example, the answer is yes: assign both A and B the value true. Using the notation $\top$ for true and $\bot$ for false, the formula becomes $\top \wedge (\top \vee \bot) = \top \wedge \top = \top$. It is possible to satisfy this boolean formula, so the answer to the SAT-problem is yes, in this case.

Not all boolean formulas can be satisfied, however. The simplest example is $A \wedge \neg A$. Since A is the only variable, we have two cases: we can give it the value true or the value false. The two cases give us the two formulas $\top \wedge \bot$ and $\bot \wedge \top$, both of which evaluates to false. For this formula, the answer to the SAT-problem is no.

In these two examples, we were able to find the answers easily by guessing, but that is obviously not possible in larger cases. One naive approach to solving the problem in the general case would be to simply test all the possible variable assignments. This is, not surprisingly, impractical: if the formula contains n variables, the number of cases becomes 2^n, and so the number of cases grows *exponentially* as the number of variables increases. In practice, the number of cases quickly becomes unreasonably large, making this approach worthless for all but the smallest formulas. We would much prefer to have an algorithm which takes polynomial time to solve the problem: that is, if n is the "size" of the problem, in our case the size of the boolean formula, then the algorithm can solve the problem in at most $f(n)$ steps, where $f(n)$ is a polynomial function. The reason polynomial time is more desirable than exponential time is that as the problem grows infinitely large, a polynomial function will always grow slower than an exponential function, so they will always be faster for larger problem instances.

In 1971 Stephen Cook proved that the SAT-problem was NP-complete, which means that if the SAT-problem could be solved in polynomial time, then so could all NP-problems. This is the main reason the SAT-problem is so well-discussed: not only is it theoretically interesting, solving it efficiently would also be of immense practical value.

It has been shown that the SAT-problem can be reduced to other problems, some similar like CNF-SAT, which is like regular SAT except that the boolean formula has to be written on a certain form. It has also been reduced to some more exotic problems like the Traveling Salesman problem, which involves finding the shortest path through a graph while visiting all the vertices. The SAT-problem has even been reduced to variations of the the popular puzzle Sudoku [15]. These problems are also NP-complete, meaning it would be of equal practical value to solve them instead of the SAT-problem, but since all these problems are shown to be NP-complete via reductions from SAT, most people still view SAT as being *the* NP-complete problem. SAT is the "grandfather" of all NP-complete problems, so to speak.

Sadly, it seems that NP-completeness, the very property which makes SAT so desirable to solve efficiently, also makes it impossible to solve efficiently: most mathematicians and computer scientists simply don't believe that NP-complete problems can be solved in sub-exponential time. Not only are there strong theoretical reasons behind this belief, there is also the fact that no sub-exponential solution has been found for any NP-complete problem, even though this is among the most studied fields in computer science. On the other hand, there is no proof for this belief: whether NP-complete problems can be solved in sub-exponential time is still an open question, after almost 40 years of dedicated research. Computer scientists would be ashamed over this, if only they had emotions.

And now that we know the stakes, it is our turn to try to solve the SAT-problem. Even though doing it efficiently is probably doomed to fail, we might still have fun along the way.

We will use an algebraic method based on Hilbert's Nullstellensatz. This work has been written with undergraduate students in mind: some algebraic background is recommended to fully understand this paper, but focus has been put on readability and compactness rather than mathematical rigor. Some concepts are introduced informally and some uninteresting concepts are skimmed over to keep things accessible. This doesn't mean that the mathematical rigor has been neglected: whenever necessary, a complete proof can be found, either in the paper itself or in a reference.

We hope that students will find the paper interesting, that programmers find it tempting to implement, and of course that professors find it thought-provoking enough to warrant a high grade. If there are any questions or comments, the author is more than willing to answer them.

2 Algebra

This chapter will mainly consist of repetition of abstract algebra. It is entirely possible to skip ahead to the section about Gröbner bases, or to skip the chapter entirely if the reader is familiar with these concepts.

2.1 Polynomial rings and ideals

2.1.1 Polynomial rings

Let k be a field. Then a **polynomial ring** $k[x]$ is the ring of single-variable polynomials with coefficients in k, and k is called the **coefficient field** of $k[x]$. Just like with ordinary polynomials, we can view the elements of the ring as functions: a polynomial $p \in k[x]$ can be **evaluated** at $a \in k$ by replacing each occurrence of the variable x in p with a. If the resulting element is b, we say that $p(a) = b$.

We can also evaluate polynomials at points outside of the field. Let $p \in k[x]$ and let k' be an extension field of k, $k \subseteq k'$. Since the elements of k are also elements of k', then the coefficients of p are also in k', meaning $p \in k'[x]$. Hence we can evaluate p at a point in k'.

We can now introduce **equations**. Let 0 denote the additive identity in the field k. Given a polynomial $p \in k[x]$, is there any element a so that $p(a) = 0$? If so, we say that a is a **solution** or a **root** to p, and that a **solves** p. All polynomials have roots, but they are not always in the original field: they may lie in an extension field. If k is a field and we are lucky enough that all polynomials $p \in k[x]$ have corresponding solutions $a \in k$, then we say that k is **algebraically closed**.

Example 1. The real numbers $\mathbb{R}$ is a field, and the complex numbers $\mathbb{C}$ is an extension field of the real numbers, $\mathbb{R} \subset \mathbb{C}$. The polynomial $x^2 + 1$ only has coefficients in $\mathbb{R}$, but it has no roots there, so $\mathbb{R}$ is not algebraically closed. The polynomial does have roots in $\mathbb{C}$, as $\pm i$ solves the polynomial. It even turns out that all polynomials in $\mathbb{C}[x]$ have solutions in $\mathbb{C}$, meaning that $\mathbb{C}$ is algebraically closed.

If we have a field k, then the smallest extension field of k which is algebraically closed is called the **algebraic closure** of k, and is denoted as $\overline{k}$. Each field has exactly one algebraic closure, unique up to isomorphism, and if a field is already algebraically closed, then it is its own algebraic closure. As it is usually more convenient do deal with fields that are algebraically closed, we often refer to an algebraic closure of a field rather than a field itself.

Example 2. Continuing our example, we have that $\mathbb{C}$ is the algebraic closure of $\mathbb{R}$.

We can also have polynomials in more than one variable: if k is a field and X is a set of variables $x_1, x_2, \ldots, x_n$, then $k[X]$, or $k[x_1, x_2, \ldots, x_n]$, is the ring of polynomials in the variables $x_1, x_2, \ldots, x_n$ with coefficients in k. A **point** is an n-tuple $(a_1, a_2, \ldots, a_n), a_i \in k'$ where k' is an extension field of k, and we can evaluate a polynomial $p \in k[x_1, x_2, \ldots, x_n]$ at the point $(a_1, a_2, \ldots, a_n)$ by replacing each occurrence of x_i in p with a_i, for $i = 1, \ldots, n$. If the resulting element is b, then we say that $p(a_1, a_2, \ldots, a_n) = b$.

We can similarly create equations, asking for points at which p becomes zero.

Polynomials in more than one variable are called **multivariate** polynomials. Polynomials in only one variable are sometimes called **univariate** polynomials.

Let k be a field, let V be a vector space over k, and let v be a finite subset of V. Then a sum $c_1 v_1 + c_2 v_2 + \ldots + c_m v_m$, where $c_i \in k$, $v_i \in v$, is called a **linear combination** of the elements in v, or a linear combination of the v_i. We can let v be a infinite subset, or even V itself, but the sum must only contain a finite number of terms. If k is not necessarily a field, but just a set of elements which can be multiplied with the elements in v, then we simply call such a sum a **k-combination** of elements in v.

Example 3. Let $\mathbb{R}[x]$ be our vector space. The set of all linear combinations of the elements in $\mathbb{R}[x]$ is the set $\{r_1p_1 + r_2p_2 + \ldots + r_mp_m : r_i \in \mathbb{R}, p_i \in \mathbb{R}[x], m \in \mathbb{N}\}$

If we let k be the polynomial ring $\mathbb{R}[y]$, and our set v be a finite subset of $\mathbb{R}[x]$, for example, $v = \{2x^2 + x + 1, x^3 - x\}$, then one possible k-combination of the elements in v is $(y^2 + y + 1)(2x^2 + x + 1) + 1 \cdot (x^3 - x)$.

Some more rigorous notation: If we have a polynomial ring $R = k[x_1, x_2, \ldots, x_n]$, then a **monomial** $m \in R$ is a product of the variables $x_1, \ldots, x_n$, that is, $m = x_1^{\alpha_1} x_1^{\alpha_2} \ldots x_n^{\alpha_n}$, where the α_i are non-negative integers. A **term** $t \in R$ is a monomial with a coefficient, $t = cm$, where m is a monomial and $c \in k$. A **polynomial** $p \in R$ is simply a sum of terms, or equivalently, a linear combination of monomials.

2.1.2 Ideals and varieties

If we have a ring[1] R, then an **ideal** I is a subset of R which is closed under addition with elements in I, and closed under multiplication with elements in R. This means that if $a, b \in I$, then $ca + db \in I$ for all $c, d \in R$. In other words, all R-combinations of elements in I belong to I as well.

If we have a ring R and a set $S = \{a_1, a_2, \ldots, a_m\}, a_i \in R$, then the set I consisting of all R-combinations of the elements in S, $I = \{b_1a_1 + b_2a_2 + \ldots + b_ma_m : b_i \in R\}$, fulfills the conditions to be an ideal. We say that I is **generated** by the elements in S, denoted $I = \langle a_1, a_2, \ldots, a_m \rangle$. The set generating an ideal does not have to be finite: ideals can be generated by infinite sets. Note that ideals can often be generated in more than one way; the same ideal can be generated by different sets. If an ideal can be generated by a finite set, it is called a **finitely generated** ideal. It turns out that all ideals of polynomial rings are finitely generated.

Note that it is not enough to define an ideal I by a set of generators: one must also specify which ring R the ideal is a subset of, to specify what kind of coefficients we are allowed to use in the sums.

Example 4. The ideal $I = \langle 2 \rangle$ can mean different sets depending on which ring we are in. If we are in Z then I is just the set of all even numbers. If we are in $\mathbb{R}[x]$, then I is actually the whole ring $\mathbb{R}[x]$ itself, since every polynomial $p \in \mathbb{R}[x]$ can be written as the product $\frac{p}{2} \cdot 2$.

We can define some operations on ideals: let $I = \langle a_1, a_2, \ldots, a_n \rangle, J = \langle b_1, b_2, \ldots, b_m \rangle$ be two arbitrary ideals of the same ring R. Then we define $I + J$ as the ideal generated by the generators of both I and J, $I + J = \langle a_1, a_2, \ldots, a_n, b_1, \ldots, b_m \rangle$. Note that this is identical to saying that $a \in I, b \in J \Rightarrow ac + bd \in I + J$ for all $c, d \in R$.

Say that we have an ideal I of a polynomial ring R, either univariate or multivariate. We can once again view the polynomials in I as equations, and ask ourselves if there are any points which solves all polynomials in I. For that reason, we introduce the concept of variety:

Definition 2.1 (Variety). *Let I be a polynomial ideal, $I \subseteq k[x_1, x_2, \ldots, x_n]$. The set of points which solve all polynomials in I is known as the* **variety** *of I, and is denoted as $V(I)$. More formally, $V(I) = \{(a_1, a_2, \ldots, a_n) : a_i \in \overline{k}, p(a_1, a_2, \ldots, a_n) = 0, \forall p \in I\}$.*

[1]For the sake of simplicity, we will assume that all rings are commutative, and that $1 \neq 0$ in them.

If we are only interested in points with coordinates in a particular field h, either a field containing k or a subfield of k, we can denote that set $V_h(I)$. This can be useful for specifying, for example, that we are interested in solutions with coordinates in the coefficient field.

A set that is a variety to some ideal is called an **algebraic set**. The smallest ideal with the variety S is denoted $I(S)$.

Theorem 2.2. *Let $R = k[x_1, \ldots, x_n]$ be a polynomial ring, and let h be either a subfield or an extension field of k. Let I be the ideal generated by $p_1, p_2, \ldots, p_m$, and let S be the set of points with coordinates in h which solve $p_1, p_2, \ldots, p_m$. Then $S = V_h(I)$. In other words, the variety of an ideal consists of the solutions of the generators of the ideal.*

Proof. As $p_1, p_2, \ldots, p_m \in I$, the elements of the variety must solve the p_i, so it is clear that $V_h(I) \subseteq S$. Let p be an arbitrary element in I and let s be an arbitrary element of S. Then p can be written as a R-combination of the generators of the ideal, $p = g_1p_1 + g_2p_2 + \ldots + g_mp_m$, and since all elements of S solve $p_1, p_2, \ldots, p_m$, we have $p(s) = g_1(s)p_1(s) + \ldots + g_m(s)p_m(s) = g_1(s) \cdot 0 + \ldots + g_m(s) \cdot 0 = 0$, so s solves p, giving $S \subseteq V_h(I)$. In total we have $S = V_h(I)$. □

Note that Theorem 2.2 is of course also true if we let h be the algebraic closure of the coefficient field; this means that all solutions to the p_i solves all polynomials in I. We will finish with some useful variety properties:

Theorem 2.3.

1. *Let I and J be two polynomial ideals, $I \subseteq J$. Then $V(J) \subseteq V(I)$.*
2. *Let I and J be two arbitrary polynomial ideals. Then $V(I + J) = V(I) \cap V(J)$.*

Proof.

1. Let a be an arbitrary element in $V(J)$. This means that for all polynomials $p \in J$ we have $p(a) = 0$. But as $I \subseteq J$, the same is true for all polynomials in I, so $a \in V(I)$ as well.
2. All elements in $V(I) \cap V(J)$ must solve all generators of both I and J. But these are exactly the generators of $I + J$, so $V(I) \cap V(J) = V(I + J)$ according to Theorem 2.2.

□

2.1.3 A word on $\mathbb{Z}_2$

We will perform many calculations in the polynomial ring $\mathbb{Z}_2[x_1, \ldots, x_n]$ in this paper. The simplest way to describe the field $\mathbb{Z}_2$ is simply as the set $\{0, 1\}$, where multiplication and division follows the usual rules, but where $1 + 1 = 0$ and $0 - 1 = 1$. This means that in the polynomial ring $\mathbb{Z}_2[x_1, \ldots, x_n]$, the only available coefficients for our terms are 0 and 1. Moreover, for arbitrary polynomials $p \in \mathbb{Z}_2[x_1, \ldots, x_n]$, we have that $p + p = (1 + 1)p = 0 \cdot p = 0$, and $0 - p = (0 - 1)p = 1 \cdot p = p$. In other words, there is no difference between plus and minus in this ring, making it very interesting to work in. From here on, if a minus sign suddenly changes into a plus sign, it was probably not an error, but rather an attempt to reduce clutter.

2.1.4 Hilbert's Nullstellensatz

One form of the famous David Hilbert's theorem Hilbert's Nullstellensatz states that

Theorem 2.4 (Hilbert's Nullstellensatz, weak form)**.** *Let I be a polynomial ideal in $k[x_1, x_2, \ldots, x_n]$. If and only if $1 \in I$, then $V(I) = \emptyset$.*

We will not prove this statement, but a proof can be found in [1]. Let's try to understand what the theorem means in practical terms. If an ideal contains 1, then obviously this means that the variety must be empty, since the constant 1 can never be "evaluated" to zero, no matter which point we consider.

But the weak form of Hilbert's Nullstellensatz tells us that the converse is true as well: when $1 \notin I$, then there must be some element in $V(I)$. In other words, there must be at least some point which solves all polynomials in the ideal. Whether or not 1 is part of the ideal or not, is equivalent to whether or not there are solutions to the elements in the ideal.

Let's say that we have a set of polynomials $p_1, p_2, \ldots, p_m \in R = k[x_1, x_2, \ldots, x_n]$, and that we wanted to know if they were solvable, that is, if there were any point a so that $p_1(a) = p_2(a) = \ldots = p_m(a) = 0$. Recall that Theorem 2.2 states that the variety of an ideal is the same as the set of points solving the generators of the ideal, so this is equivalent to asking if $V(\langle p_1, p_2, \ldots, p_m\rangle)$ is non-empty. Hilbert's Nullstellensatz now states that the variety is non-empty if and only if the ideal does not contain 1. In other words, a set of polynomials $p_1, p_2, \ldots, p_m$ is unsolvable if and only if 1 can be written as a R-combination of the p_i.

This can be a very powerful tool, but there is one catch: there doesn't seem to be an easy way of checking whether or not a given polynomial (in our case, 1) is part of an ideal or not. That problem will be discussed in the next section.

2.2 Gröbner bases

2.2.1 Introduction to Reduction

Remember integer division? How, for each pair of integers a, b, where $b \neq 0$, there exists unique integers q, r with $r = 0$ or $r < b$, so that $a = bq + r$? Similarly one can divide univariate polynomials: for each pair of polynomials $f, p \in k[x]$, where $p \neq 0$, there exists unique polynomials $q, r \in k[x]$, where $\deg(r) < \deg(p)$ or $r = 0$, so that $f = qp + r$. As in the integer case, we call q the **quotient** and r the **remainder**.

If two polynomials f_1 and f_2 yield the same remainder when dividing by p, we say that $f_1 \equiv f_2 \bmod p$, similar to the integer case. Another way of denoting this is $f_1 \xrightarrow{p} f_2$, or that f_1 can be **reduced** with p to f_2. Note that if $f_1 \xrightarrow{p} f_2$, then $f_1 = f_2 + qp$ for some polynomial q.

We can also use reduction in a more active sense: we say that we reduce a polynomial f with p when we simply subtract a multiple of p, so $f \xrightarrow{p} f - qp$ for any polynomial q. Unless otherwise noted, reducing f with p in this way means picking q in such a way as to minimize the degree of $f - qp$. This actually turns out to coincide with polynomial division: the $f - qp$ with smallest possible degree is in fact the remainder r when dividing f with p. For that reason we will sometimes refer to the remainder r when dividing f with p, by saying f reduced with p. Don't forget, however, that reduction arrows work slightly differently: $f_1 \xrightarrow{p} f_2$ simply means that there *exists* some q so that $f_1 = f_2 - qp$, not that this q necessarily yields the smallest degree.

Reduction with a fixed polynomial p is obviously transitive: if $f_1 \xrightarrow{p} f_2 \xrightarrow{p} f_3$, then $f_1 \xrightarrow{p} f_3$.

Finally, since it is often more convenient (both for authors and readers) to deal with monic polynomials[2], we will cheat a little: in this paper, we will always multiply the final result of a polynomial reduction with the inverse of the leading coefficient, so as to make the polynomial monic. This is a common convention when dealing with this type of reductions, as it reduces clutter while preserving the structure of the polynomial. This does not affect the uniqueness of quotients or remainders.

Having refreshed the subject of polynomial reduction, let's turn our eyes to an application...

2.2.2 Deciding ideal membership

Say that we have a polynomial ideal I and a polynomial f, and we wanted to find out whether or not $f \in I$. How would we do that?

Let us start with simple cases and work our way up. Say that I was generated by one univariate polynomial, $I = \langle p \rangle$, $p \in k[x]$. In this case we can use ordinary polynomial reduction: To check if $f \in I = \langle p \rangle$, we reduce f with p: If $f \xrightarrow{p} 0$, then f is a multiple of p and so $f \in I$. If reducing f with p yields a non-zero remainder, then due to the uniqueness of remainders f is not a multiple of p, and is not in I.

A slightly more interesting case is when I is generated by several single-variable polynomials, $I = \langle p_1, p_2, \ldots, p_m \rangle$, $p_i \in k[x]$. Not much more interesting, however: in this case we have that I can also be described as $I = \langle \gcd(p_1, p_2, \ldots, p_m) \rangle$, where gcd denotes the greatest common divisor. That this works is partly because the greatest common divisor of univariate polynomials can be written as a $k[x]$-combination of the polynomials, so the greatest common divisor is itself part of the ideal. Using the Euclidean algorithm for polynomials [14], we can find this greatest common divisor, and then treat I as if generated by one polynomial, as in the first case.

If I is generated by one multivariate polynomial, $I = \langle p \rangle$, $p \in k[x_1, x_2, \ldots, x_n]$, we start to reach problems. It is tempting to try and generalize the polynomial reduction method to several variables, but doing so will prove more difficult than one might expect, as the next section will show us...

2.2.3 Term orderings

Say that we wanted to reduce the polynomial f with the polynomial p. If f and p were univariate polynomials, the first step would be to identify the leading terms of f and p, by finding the terms of highest degree. If f and p were multivariate polynomials, we would also try to find the leading terms, but this is where things get tricky: what is the leading term in a multivariate polynomial? For example, what is the leading term in the polynomial $x_1^2 + x_1x_2^2$?

The degree of a multivariate monomial is defined as a the sum of the exponents on the variables, and one could argue that $x_1x_2^2$ should be the leading term, since it has the highest degree. But one could also argue that x_1^2 should be the leading term, if one considers the exponent of x_1 more important than the one of x_2, and thinks x_1 should have higher priority in deciding the order. If we introduce the notation $\succ$ meaning "higher order than" and $\prec$ meaning "lower order than", then our first way of reasoning gives us $x_1x_2^2 \succ x_1^2$ and the second one $x_1x_2^2 \prec x_1^2$.

[2]Polynomials with 1 as the leading coefficient.

Both of the above orderings seem natural and intuitive, and they are consistent with the univariate case. However, attempting multivariate polynomial division with them gives wildly different results depending on which one is used: the quotient and remainder are no longer unique, but depend on which monomial ordering we decide on. Neither monomial ordering can be shown to be more "correct" than the other: they are both consistent and give meaningful results. In fact, these aren't the only ways of ordering monomials: many more exist, though not all are useful. We introduce the concept of **admissible orderings** to give us some criteria for monomial orderings with useful properties.

Definition 2.5 (Admissible monomial ordering). *A relation $\succ$ between monomials is an* **admissible monomial ordering** *if it is*

1. *a* ***total ordering****: for all monomials m_1, m_2 we have $m_1 \succ m_2$, $m_1 \prec m_2$ or $m_1 = m_2$.*
2. ***transitive****: if $m_1 \succ m_2$ and $m_2 \succ m_3$, then $m_1 \succ m_3$.*
3. ***compatible with monomial multiplication****: if $m_1 \succ m_2$, then $mm_1 \succ mm_2$ for all monomials m.*
4. *$1 \prec m$ for all monomials $m \neq 1$.*

If we have an admissible monomial ordering $\succ$ we can simply define the leading term of a polynomial as the term of highest order. We denote the leading monomial of a polynomial p as $\mathrm{lm}(p)$ and leading term as $\mathrm{lt}(p)$. Note that the difference between the two is that the leading term includes the coefficient, whereas the leading monomial is just a product of variables. For convenience we also denote the "tail" of a polynomial, the polynomial with the leading term removed, as $\mathrm{t}(p)$.

The two examples of orderings we looked at above are called **lexicographical** and **degree-lexicographical** (or lex and deglex for short), and are more rigidly defined as:

Definition 2.6 (Lex). *$m_1 = x_1^{\alpha_1}x_2^{\alpha_2}\ldots x_n^{\alpha_n} \succ m_2 = x_1^{\beta_1}x_2^{\beta_2}\ldots x_n^{\beta_n}$ if for some k we have $\alpha_1 = \beta_1, \alpha_2 = \beta_2, \ldots, \alpha_{k-1} = \beta_{k-1}, \alpha_k > \beta_k$.*

Definition 2.7 (Deglex). *$m_1 = x_1^{\alpha_1}x_2^{\alpha_2}\ldots x_n^{\alpha_n} \succ m_2 = x_1^{\beta_1}x_2^{\beta_2}\ldots x_n^{\beta_n}$ if $\deg(m_1) = \alpha_1+\alpha_2+\ldots+\alpha_n > \deg(m_2) = \beta_1+\beta_2+\ldots+\beta_n$ or if $\deg(m_1) = \deg(m_2)$ and $m_1 \succ m_2$ using lexicographical ordering.*

Less formally, lex means that the higher term has a higher exponent on the first variable where the exponent differs. In deglex, the higher term is the one with higher total degree, and if the degree is equal, lex is used to decide.

Example 5. Let p be the polynomial $x_1^2+x_1x_3^2+x_1x_2$. If we use lexicographical ordering, the terms are ordered as $x_1^2 \succ x_1x_2 \succ x_1x_3^2$, but if we use degree-lexicographical ordering, we get $x_1x_3^2 \succ x_1^2 \succ x_1x_2$.

A third ordering, not as intuitive but more useful for our purposes, is the **degree-reverse-lexicographical ordering**, degrevlex:

Definition 2.8 (Degrevlex). *$m_1 = x_1^{\alpha_1}x_2^{\alpha_2}\ldots x_n^{\alpha_n} \succ m_2 = x_1^{\beta_1}x_2^{\beta_2}\ldots x_n^{\beta_n}$ if $\deg(m_1) > \deg(m_2)$, or if $\deg(m_1) = \deg(m_2)$ and for some k we have*
$\alpha_n = \beta_n$, $\alpha_{n-1} = \beta_{n-1}, \ldots, \alpha_k < \beta_k$.

Less formally, degrevlex first compares the degree. If this is equal, it then does a lexicographical check, starting from the least important variable: the first monomial to have a lower exponent on the less important variables, is the monomial with higher order. Since the total degree is equal, this lower exponent on the less important variables, ensures that there is some more important variable with higher exponent.

This may seem like a strange criteria, but it is an admissible ordering and some algorithms, including one we will introduce later, works best with degrevlex.

Example 6. The terms $t_1 = x_1^2 x_2 x_3^2$ and $t_2 = x_1 x_2^3 x_3$ are ordered as $t_1 \succ t_2$ in both lex and deglex, but in degrevlex they are ordered as $t_1 \prec t_2$.

Note that even though we are interested in ordering the terms of a polynomial, we do not care about the coefficients: only the underlying monomial is actually interesting. For this reason, we do not distinguish between the concepts of monomial ordering and term ordering. It is also for that reason that term orderings are independent of coefficient field of the polynomial ring.

As a final remark, note that in the above examples, we implicitly assumed an order on the variables: in lex, we assume that the exponent on x_1 is more important than the exponent of x_2, that the exponent of x_2 is more important than the exponent of x_3 and so on, forming a variable ordering $x_1 \succ x_2 \succ \ldots \succ x_n$. This seems natural, but as the indexes of the variables are arbitrary, there is nothing stopping us from reordering the priority of the variables.

Example 7. Let $m_1 = x_1^2$ and $m_2 = x_1 x_2^2$. If we use lex and the regular variable ordering $x_1 \succ x_2$, then the leading term is m_1, as the exponent of x_1 is higher in m_1 than in m_2. But if we had instead used the variable ordering $x_2 \succ x_1$, the leading term would be m_2, as the exponent of x_2 now has higher priority.

As long as we are consistent, we can pick not only the term ordering, but also the variable ordering. Note though that it is not often changing the variable ordering is useful, whereas different term orderings often make a huge difference. The variable ordering used is often seen as a part of the term ordering, and if not mentioned is assumed to be the regular $x_1 \succ x_2 \succ \ldots \succ x_n$.

2.2.4 Multivariate polynomial reductions

Having overcome the problem of leading terms, we can look back at multivariate polynomial reduction. We would like to have some algorithm which, for polynomials f and p gives us a polynomial r so that $f = pq + r$ for some polynomial q, and so that r has terms of as low degree as possible, similar to reduction in the univariate case. Unfortunately it is not possible to find an algorithm which perfectly resembles the algorithm for the univariate case, since the leading terms no longer are unique. However, if we fix an ordering, we can create a similar algorithm where we reduce a polynomial f so as to receive a polynomial r, where we attempt to lower the order of the terms in r as much as possible.

The algorithm is as follows: Let $f, p \in k[x_1, x_2, \ldots, x_n]$, and let $\succ$ be an admissible monomial ordering. Let the leading term, with respect to $\succ$, of p be $\mathrm{lt}(p)$ and let S be the set of terms in f which are divisible by $\mathrm{lt}(p)$. If S is empty, then $r = f$ (No further reduction is possible). If S is not empty, then let t be the term in S of highest order, with respect to $\succ$. Perform the reduction $f_1 = f - \frac{t}{\mathrm{lt}(p)} p$. Note that the term t will be canceled out. Repeat the process with f_1 and p for as long as possible, in each step either ending the algorithm, or creating a new polynomial f_i by removing a term divisible by $\mathrm{lt}(p)$.

Algorithm 1 Multivariate polynomial reduction

```
Input: Two polynomials $f, p \in k[x_x, x_2, \ldots, x_n], p \neq 0$, and one admissible polynomial ordering $\succ$.
Output: $f$ reduced with $p$.
  $r \leftarrow f$
  Let $S$ be the set of terms in $r$ divisible by $\mathrm{lt}(p)$
  while $r \neq 0$ and $S \neq \emptyset$ do
    $t \leftarrow$ the element in $S$ of highest order.
    $r = r - \frac{t}{\mathrm{lt}(p)}p$
    Let $S$ be the set of terms in $r$ divisible by $\mathrm{lt}(p)$
  end while
  return $r$
```

We can use reduction arrows, exactly like in the univariate case. The algorithm then creates a chain of reductions $f \xrightarrow{p} f_1 \xrightarrow{p} \ldots \xrightarrow{p} r$. As we want to simplify the calculations, we also make each f_i monic.

Example 8. Let $f = 2x_1^2 + x_1x_2 + x_2$, and let $p = x_1 + x_2$. Using deglex, we have that $\mathrm{lt}(p) = x_1$, so $S = \{2x_1^2, x_1x_2\}$. The highest order element in S is $2x_1^2$, which is our t, so $f_1 = f - \frac{t}{\mathrm{lt}(p)}p = 2x_1^2 + x_1x_2 + x_2 - \frac{2x_1^2}{x_1}(x_1 + x_2) = -x_1x_2 + x_2$. Making the polynomial monic gives us $f_1 = x_1x_2 - x_2$.

The new S only contains $-x_1x_2$, so we get $f_2 = x_1x_2 - x_2 - x_2(x_1 + x_2) = -x_2^2 - x_2$, and the corresponding monic polynomial is $x_2^2 + x_2$. This time we get that S is empty, so the algorithm stops.

In the end, we have that $2x_1^2 + x_1x_2 + x_2$ reduced with $x_1 + x_2$ is $x_2^2 + x_2$.

Note the strategy of the algorithm: it tries to find a removable term with as high order as possible in f, and then reduce it in such a way that the term disappears. Other terms may appear, but they are of lower order, so in total the algorithm have reduced the order of the terms. We could conceivably reduce the polynomial in such a way that terms in f divisible by non-leading terms in p disappear, but then we would most likely add other terms of higher order.

We finally have a division algorithm for multivariate polynomials. As long as we have some fix monomial ordering, this algorithm will be able to fully reduce one polynomial with another, resulting in something similar to a remainder. Note though that this remainder will not be unique: it will depend on the monomial ordering we decided upon. However, luckily for us, there is one case where the remainder is definitely unique:

Lemma 2.9. *Let f, p be two polynomials so that $f = qp$ for some some polynomial q. Then f reduced with p will always become zero, no matter which monomial ordering we decide on.*

Proof. The proof will be an induction proof over the number of terms in q.

Say that q consists of a single term t_1, so $f = t_1p$. Then $\mathrm{lt}(f) = \mathrm{lt}(t_1p) = t_1\mathrm{lt}(p)$, no matter which ordering we use, so the initial reduction becomes $f - \frac{t_1\mathrm{lt}(p)}{\mathrm{lt}(p)}p = f - t_1p = 0$. The algorithm will yield 0 as the remainder in this case.

Next, assume that $f = qp$ implies that the algorithm will yield zero when q contains k terms. What happens when q contains $k + 1$ terms?

We have that $q = \mathrm{lt}(q) + \mathrm{t}(q)$, where $\mathrm{t}(q)$ contains k terms. We also have $\mathrm{lt}(f) = \mathrm{lt}(\mathrm{lt}(q)p + \mathrm{t}(q)p) = \mathrm{lt}(\mathrm{lt}(q)\mathrm{lt}(p) + \mathrm{lt}(q)\mathrm{t}(p) + \mathrm{lt}(p)\mathrm{t}(q) + \mathrm{t}(p)\mathrm{t}(q)) = \mathrm{lt}(q)\mathrm{lt}(p)$, no matter which ordering is used. The first reduction step in the algorithm then becomes $f - \frac{\mathrm{lt}(q)\mathrm{lt}(p)}{\mathrm{lt}(p)}p = pq - \mathrm{lt}(q)p = p\mathrm{t}(q)$. The next step in the reduction chain is to reduce $p\mathrm{t}(q)$ with p, but according to the induction assumption, this reduction will lead to zero. $\square$

The converse is also true: if we are reducing a polynomial f with p, and f is NOT a multiple of p, then the end result will never be zero.

Having shown these properties of the multivariate remainder, we can use it for our original problem: deciding ideal membership. If we wish to know if $f \in k[x_1, x_2, \ldots, x_n]$ belongs to an ideal generated by a single multivariate polynomial, $I = \langle p \rangle$, $p \in k[x_1, x_2, \ldots, x_n]$, we simply reduce f with p. If the end result is zero, p is in the ideal, otherwise not.

We have almost solved the problem of deciding ideal membership completely, but the most difficult step remains: what do we do if our ideal is generated by several multivariate polynomials, $I = \langle p_1, p_2, \ldots p_m \rangle$, $p_i \in k[x_1, x_2, \ldots, x_n]$? Once again, it is tempting to generalize the method used in the univariate case, where we used the fact that the ideal was also generated by the greatest common divisor of the p_i. This was true partly because greatest common divisors could be written as $k[x]$-combinations of the original polynomials in the univariate case.

This is sadly not true for multivariate polynomials. For example, in $\mathbb{Z}_2[x, y]$, the only common divisor x and y has is 1, so that is also the greatest common divisor. But 1 can clearly not be written as any sort of combination of x and y, so 1 is not in the ideal generated by x and y! We must find another method for determining ideal membership when dealing with several multivariate polynomials. Perhaps we can expand our division algorithm to somehow perform reductions with multiple polynomials?

So let us do that: say that we have a polynomial f and several polynomials $p_1, p_2, \ldots, p_m$. We wish to find a polynomial r such that $f = q_1p_1 + q_2p_2 + \ldots + q_mp_m + r$ for some polynomials q_i, so that the terms in r are of as low order as possible.

Let $f, p_1, \ldots, p_m \in R = k[x_1, x_2, \ldots, x_n]$, and let $\succ$ be an admissible polynomial ordering. Let S_i be the set of terms in f which are divisible by $\mathrm{lt}(p_i)$ for $i = 1, \ldots, m$. If all the S_i are empty, then $r = f$. Otherwise, let S_j be the non-empty set S_i with lowest index, and let t be the term in S_j of highest order. Finally, let $f_1 = f - (t/\mathrm{lt}(p_j))p_j$. The term t will be canceled out. Repeat the process with f_1 and the p_i for as long as possible, in each step either ending the algorithm, or removing a term divisible by any of the $\mathrm{lt}(p_i)$, giving higher priority to reducing with p_i's with lower index.

Some notation is in order: if r is the remainder when reducing f with $p_1, p_2, \ldots, p_m$, then we say that $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m)) = r$, and in this paper, we will make r monic. Note that $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m))$ implies an ordering on the p_i - we call this the **reduction ordering**.

Example 9. Let $f = x_1^2x_2, p_1 = x_1x_2 - x_2^2, p_2 = x_1^2$. Using deglex, what is $\mathrm{rem}(f, (p_1, p_2))$?

We have that $\mathrm{lt}(p_1) = x_1x_2$, $\mathrm{lt}(p_2) = x_1^2$, so $S_1 = S_2 = \{x_1^2x_2\}$. Then $f_1 = x_1^2x_2 - \frac{x_1^2x_2}{x_1x_2}(x_1x_2 - x_2^2) = x_1x_2^2$. As this already is monic, we do not alter it.

Next we get $S_1 = \{x_1x_2^2\}$, $S_2 = \emptyset$, so $f_2 = x_1x_2^2 - \frac{x_1x_2^2}{x_1x_2}(x_1x_2 - x_2^2) = x_2^3$, which handily enough is monic. In the next step, we get that $S_1 = S_2 = \emptyset$, so we are done: the remainder is x_2^3.

Keep in mind that if $f \xrightarrow{p} r$, then f is a R-combination of r and p, since $f = r + pq$ for some q. This can be expanded to several polynomials: we have that f is

Algorithm 2 Polynomial reduction with several polynomials

```
Input: A polynomial $f$ and a ordered set of polynomials $p_i$ in $k[x_x, x_2, \ldots, x_n], p_i \neq 0$,
  and one admissible polynomial ordering $\succ$.
Output: $f$ reduced with the $p_i$.
  $r \leftarrow f$
  $m \leftarrow$ the number of $p_i$
  while $r \neq 0$ do
    for $i = 1$ to $m$ do
      let $S_i$ be the set of terms in $r$ divisible by $\mathrm{lt}(p_i)$
      if $S_i$ is non-empty then
        let $t$ be the element in $S$ with highest order, remember $i$ and break the for-loop.
      end if
    end for
    if some $S_i$ was non-empty, the loop found a $t$ then
      $r = r - \frac{t}{\mathrm{lt}(p_i)} p_i$
    else
      all the $S_i$ were found empty, break the while-loop
    end if
  end while
  return $r$
```

a R-combination of $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m))$ and the $p_1, p_2, \ldots, p_m$. This means that if $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m)) \in I$ and $p_1, p_2, \ldots, p_m \in I$, then $f \in I$ as well.

Of particular interested is when the remainder is zero: for example if $f \xrightarrow{p} 0$. That means that f is simply a multiple of p, as discussed earlier. Similarly for multiple polynomials: if $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m)) = 0$, then f is a R-combination of the p_i, and $p_1, p_2, \ldots, p_m \in I$ implies $f \in I$.

Can this method be used to determine ideal membership then? Unfortunately, not as it is. It actually turns out that the final remainder is not unique, but depends on the chosen reduction ordering, or how we have ordered the p_i in $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m))$. Let's try the above example again, but having ordered the p_i differently...

Example 10. This time we calculate $\mathrm{rem}(f, (p_2, p_1))$. Once again we have $S_2 = S_1 = \{x_1^2 x_2\}$. As we are using another reduction ordering, however, we get $f_1 = x_1^2 x_2 - \frac{x_1^2 x_2}{x_1^2}(x_1^2) = 0$.

Note that the remainder became zero in this case: this means that f is a R-combination of the p_i, so $f \in \langle p_1, p_2 \rangle$. But our first attempt yielded a non-zero remainder! The reduction method can clearly give false negatives: we may get a non-zero remainder, even if the polynomial actually is a R-combination of the divisors. It seems we can not use this method to decide ideal membership.

Note there are two different orderings affecting the outcome of the reductions here - first we have the monomial ordering (lex, deglex, degrevlex, etc.) which decides whether, for example, $x_1 x_2^2 \succ x_1^2$ or $x_1 x_2^2 \prec x_1^2$. When performing reductions, we usually have a fix monomial ordering, set from the start. The second type of ordering we speak of is the reduction ordering, how we have ordered the p_i in $\mathrm{rem}(f, (p_1, p_2, \ldots, p_m))$.

2.2.5 Gröbner Bases

This is where Gröbner bases enter: A **Gröbner basis** G is a set of multivariate polynomials generating an ideal I, with the incredibly convenient property that, for a fix monomial ordering $\succ$, if we try to reduce a polynomial f with the polynomials in G, the reduction ordering doesn't matter: the final remainder only depends on the ideal itself, not on how we've ordered the generators we represent it with. More importantly, if f is part of the ideal generated by G, then this remainder will always be zero. Gröbner bases, are, in a sense, multivariate generalizations of greatest common divisors.

Example 11. Let our ring be $R = \mathbb{R}[x_1, x_2]$, and let $p_1 = x_1x_2 + x_1$, $p_2 = x_1x_2 - x_2$, $f = x_1^2 - x_1x_2^2$ be elements in $\mathbb{R}[x_1, x_2]$, and form the ideal $I = \langle p_1, p_2\rangle$. Using lex, we can calculate some remainders. For the sake of convenience, we simplify the notation by using reduction arrows.

rem$(f, (p_1, p_2))$: $x_1^2 - x_1x_2^2 \xrightarrow{p_1} x_1^2 + x_1x_2 \xrightarrow{p_1} x_1^2 - x_1$

rem$(f, (p_2, p_1))$: $x_1^2 - x_1x_2^2 \xrightarrow{p_2} x_1^2 - x_2^2$

Note that the two remainders are different, and can not be reduced any further either with p_1 or p_2. As we got non-zero remainders, we can not conclude whether or not $f \in I$.

Luckily, I can also be generated by the two polynomials $p_3 = x_1 + x_2$, $p_4 = x_2^2 + x_2$, $I = \langle p_3, p_4\rangle$. This can easily be verified by checking that both p_1 and p_2 can be written as R-combinations of p_3 and p_4, and vice versa. In our case, we have that $p_1 = (x_2 + 1)p_3 - p_4$, $p_2 = x_2p_3 - p_4$, $p_3 = p_1 - p_2$, $p_4 = x_2p_1 + (-x_2 - 1)p_2$.

And we are even so fortunate that $\{p_3, p_4\}$ is a Gröbner basis: this time the reductions give us

rem$(f, (p_3, p_4))$: $x_1^2 - x_1x_2^2 \xrightarrow{p_3} x_1^2 + x_2^3 \xrightarrow{p_3} x_1x_2 - x_2^3 \xrightarrow{p_3} x_2^3 + x_2^2 \xrightarrow{p_4} 0$

rem$(f, (p_4, p_3))$: $x_1^2 - x_1x_2^2 \xrightarrow{p_4} x_1^2 + x_1x_2 \xrightarrow{p_3} 0$

Both reductions give the same remainder, regardless of how the p_i are ordered. And this remainder happens to be zero, so we can even say that $f \in I$. Had the remainder been non-zero, we could with confidence have said that $f \notin I$.

A more strict definition of Gröbner basis is:

Definition 2.10 (Gröbner basis)**.** *Let $k[x_1, \ldots, x_n]$ be a polynomial ring. If I is an ideal of $k[x_1, \ldots, x_n]$, then we define $l(I)$ to be the ideal generated by the leading monomials of the elements in I: $l(I) = \langle lm(f) : f \in I\rangle$. A set $G = \{g_1, g_2, \ldots, g_n\}$ generating I is called a* **Gröbner basis** *for I if $\langle lm(g_1), lm(g_2), \ldots, lm(g_n)\rangle = l(I)$.*

Proving that reduction with a a Gröbner basis yields a unique remainder is nontrivial, and will not be done here [1]. All ideals have at least one Gröbner basis, and they can be found using the Buchberger algorithm, described in Section 2.3.

A Gröbner basis $G = \{p_1, p_2, \ldots, p_m\}$ is called **reduced** if no leading monomials divide any monomial in any other p_i, and if all leading coefficients are one. Also, a Gröbner basis is called **minimal** if it no longer is a Gröbner basis for the same ideal when removing any element from the basis. It turns out that reduced minimal Gröbner bases are unique: for each ideal and monomial ordering, there is precisely one reduced minimal Gröbner basis. This uniqueness makes reduced minimal Gröbner bases very useful, but as they are not necessary for this paper, they will not be discussed further.

Example 12. In example 11, we have that $\{x_1+x_2, x_2^2-x_2\}$ is actually a reduced minimal Gröbner basis.

Since reducing a polynomial f with the polynomials in a Gröbner basis G gives a unique remainder independent of how the elements in G are ordered, we can expand on our previous notation: we say that $\mathrm{rem}(f, G) = r$ if r is the remainder when reducing f with the elements in G, independent of how they are ordered. Since this r only depends on I and f, we can call r the **normal form** of f with respect to I.

Example 13. Continuing example 11, the normal form of $x_1^2 - x_1x_2$ with respect to $I = \langle x_1x_2 + x_1, x_1x_2 - x_2\rangle$ is zero, as seen in Example 11. The normal form of x_1x_2 is x_2, as can be seen through the following reduction chain, where we reduce with the Gröbner basis: $x_1x_2 \xrightarrow{p_3} x_1x_2 - x_2(x_1 + x_2) = x_2^2 \xrightarrow{p_4} x_2^2 - (x_2^2 + x_2) = x_2$.

Having overcome the problem of non-unique remainders, we can use the reduction technique to easily decide whether or not a polynomial f is in an ideal I: first find a Gröbner basis G of I. If and only if $\mathrm{rem}(f, G) = 0$ then $p \in I$.

Particularly interesting to us are Gröbner bases of ideals consisting of the whole polynomial ring. First note that an ideal $I \subseteq R = k[x_1, x_2, \dots, x_n]$ consists of the entire polynomial ring R if and only if $1 \in I$: if the ideal contains every element, then surely it contains 1. Similarly, if it contains 1, then it must also contain every other element $m \in R$, since $m = m \cdot 1$. If we know that G is a Gröbner basis to an ideal I which consists of the entire ring, what does that tell us about G?

Theorem 2.11. *Let G be a Gröbner basis of the ideal $I \subseteq k[x_1, x_2, \dots, x_n]$. Then $I = k[x_1, x_2, \dots, x_n]$ if an only if G contains the element 1.*

Proof. Remember that $1 \in I$ means that $\mathrm{rem}(1, G) = 0$, since G is a Gröbner basis. Note that if we reduce 1 with a polynomial $p_i \in G$, the result will either be 1 or 0, as 1 is the monomial of lowest term order. This means the reduction process must have contained one reduction on the form $1 \xrightarrow{p} 0$ for some polynomial $p \in G$. But in order for the remainder to be 0, the leading term of p must divide 1. Since none of the variables $x_1, x_2, \dots, x_n$ divides 1, the leading term of p may not contain any variables: the leading monomial of p is 1. This in turn means that the leading term of p must be 1, as we only consider monic polynomials. Since all other monomials have higher order than 1, p can not contain any other terms: we must have that $p = 1$.

Conversely, if G contains 1, then clearly G generates the entire ring. □

2.3 The Buchberger algorithm

The Buchberger algorithm is the algorithm used for finding a Gröbner basis for an ideal I, if we know one set generating I. Before we can discuss the Buchberger algorithm, we will need to introduce S-polynomials.

Definition 2.12 (S-polynomial)**.** *Given two polynomials p_i and p_j, we define the* **S-polynomial** *of p_i and p_j, or $S_{i,j}$, as*

$$S_{1,2} = \frac{\mathrm{lcm}(\mathrm{lt}(p_i), \mathrm{lt}(p_j))}{\mathrm{lt}(p_i)} p_i - \frac{\mathrm{lcm}(\mathrm{lt}(p_i), \mathrm{lt}(p_j))}{\mathrm{lt}(p_j)} p_j \tag{1}$$

where $\mathrm{lcm}(a, b)$ is the least common multiple of a, b.

Note that $S_{i,j}$ is a R-combination of p_i and p_j, meaning that if p_i and p_j belong to an ideal, then so does their S-polynomial.

And now, the Buchberger algorithm. We fix some admissible monomial ordering, and start with a polynomial ideal I generated by the n polynomials in the set $G = (p_1, p_2, \dots, p_n)$. Let L be the set of all the non-ordered integer pairs from 1 to n, not including pairs of the same integer. Initially, we have

$$L = \{(1, 2), (1, 3), \dots, (1, n - 1)(1, n), (2, 3), (2, 4), \dots, (n - 1, n)\}$$

Remove an element $(i, j) \in L$, and calculate $S_{i,j}$. Let $r_{i,j}$ be the remainder when reducing $S_{i,j}$ with the elements of G, in any order. If $r_{i,j} = 0$, do nothing. Otherwise, if G contains m elements, let $p_{m+1} = r_{i,j}$, add p_{m+1} to G, and add to L the integer pairs $(1, m + 1), (2, m + 1), \dots, (m, m + 1)$.

Repeat the process as long as there are elements $(i, j) \in L$: calculate $r_{i,j}$, and if it is non-zero, add it to G and add all pairs containing it to L. When L is empty, G will be a Gröbner basis for I.

Algorithm 3 The Buchberger algorithm

```
Input: A set $G$ of polynomials $p_i$ generating the ideal $I$.
Output: A Gröbner basis $G$ generating the ideal $I$.
  $n \leftarrow$ the number of elements in $G$
  Let L be an empty set
  for $i = 1$ to $n - 1$ do
    for $j = i + 1$ to $n$ do
      Add $(i, j)$ to L
    end for
  end for
  while $L$ is not empty do
    Pick an element $(a, b)$ from $L$
    $r \leftarrow \text{rem}(S_{a,b}, G)$ (Where the $p_i$ in $G$ have some arbitrary ordering)
    if $r = 0$ then
      do nothing
    else
      $p_{n+1} \leftarrow R$
      Add $p_{n+1}$ to $G$
      for $i = 1$ to $n$ do
        Add $(i, n + 1)$ to $L$
      end for
      $n = n + 1$
    end if
  end while
  return $G$
```

Proving that the result is a Gröbner basis is not simple and will not be done here [1], but it is at least clear that the ideals generated are the same: we never remove polynomials from G, and each $r_{i,j} \in I$, $S_{i,j} \in I$, since it is an S-polynomial reduced with elements in I.

Example 14. Let $p_1 = x_1x_2 + x_1$, $p_2 = x_1x_2 - x_2$, let $I = \langle p_1, p_2 \rangle \subseteq \mathbb{R}[x_1, x_2]$. Using lex as our ordering, we can use the Buchberger algorithm to find a Gröbner basis for I.

Initially we have $G = \{x_1x_2+x_2, x_1x_2-x_2\}$, $L = \{(1,2)\}$, so we begin by calculating $S_{1,2}$.

$$\begin{aligned}S_{1,2} =& \frac{\operatorname{lcm}(\operatorname{lt}(p_1), \operatorname{lt}(p_2))}{\operatorname{lt}(p_1)}p_1 - \frac{\operatorname{lcm}(\operatorname{lt}(p_1), \operatorname{lt}(p_2))}{\operatorname{lt}(p_2)}p_2 =\\ =&\frac{x_1x_2}{x_1x_2}(x_1x_2+x_1) - \frac{x_1x_2}{x_1x_2}(x_1x_2-x_2) = (x_1x_2+x_1)-(x_1x_2-x_2) =\\ =&x_1+x_2\end{aligned}$$

We then calculate $\operatorname{rem}(S_{1,2}, (p_1, p_2))$, but as the leading terms of p_1 and p_2 does not divide any of the terms in $S_{1,2}$, we can immediately conclude that the remainder will be $S_{1,2} = x_1 + x_2$ itself. As the remainder is nonzero, we let $p_3 = x_1 + x_2$, and add it to our basis. We also add $(1, 3), (2, 3)$ to L, so we now have $L = \{(1, 3), (2, 3)\}$, $G = \{x_1x_2 + x_1, x_1x_2 - x_2, x_1 + x_2\}$

Next we pick $(1, 3)$ from L, and calculate $S_{1,3}$

$$\begin{aligned}S_{1,3} =& \frac{\operatorname{lcm}(x_1x_2, x_1)}{x_1x_2}(x_1x_2+x_1) - \frac{\operatorname{lcm}(x_1x_2, x_1)}{x_1}(x_1+x_2) =\\ =&\frac{x_1x_2}{x_1x_2}(x_1x_2+x_1) - \frac{x_1x_2}{x_1}(x_1+x_2) = (x_1x_2+x_1) - x_2(x_1+x_2) =\\ =&x_1 - x_2^2\end{aligned}$$

Then we calculate the remainder $\operatorname{rem}(S_{1,3}, (p_1, p_2, p_3)) = \operatorname{rem}(x_1 - x_2^2, (x_1x_2 + x_1, x_1x_2 - x_2, x_1 + x_2))$:

$$x_1 - x_2^2 \xrightarrow{p_3} x_1 - x_2^2 - (x_1 + x_2) = -x_2^2 - x_2$$

The corresponding monic polynomial is $x_2^2 + x_2$, and as we once again receive a nonzero remainder, we call it p_4 and add it to G. Now add $(1, 4), (2, 4), (3, 4)$ to L, so $L = \{(2, 3), (1, 4), (2, 4), (3, 4)\}$, and $G = \{x_1x_2 + x_1, x_1x_2 - x_2, x_1 + x_2, x_2^2 + x_2\}$

We next pick $(2, 4)$ and calculate the S-polynomial.

$$\begin{aligned}S_{2,4} =& \frac{x_1x_2^2}{x_1x_2}(x_1x_2-x_2) - \frac{x_1x_2^2}{x_2^2}(x_2^2+x_2) = x_2(x_1x_2-x_2) - x_1(x_2^2+x_2) =\\ =& -x_1x_2 - x_2^2\end{aligned}$$

And the remainder, $\operatorname{rem}(-x_1x_2 - x_2^2, (x_1x_2 + x_1, x_1x_2 - x_2, x_1 + x_2, x_2^2 + x_2))$...

$$\begin{aligned}&-x_1x_2 - x_2^2 \xrightarrow{p_1} -x_1x_2 - x_2^2 + (x_1x_2 + x_1) =\\ =&x_1 - x_2^2 \xrightarrow{p_3} x_1 - x_2^2 - (x_1 + x_2) = x_2^2 + x_2 \xrightarrow{p_4} -x_2^2 - x_2 + (x_2^2 + x_2) = 0\end{aligned}$$

As the remainder becomes zero, we add nothing to either our basis, or to L.

Our next element from L is $(2, 3)$. First the S-polynomial...

$$S_{2,3} = \frac{x_1x_2}{x_1x_2}(x_1x_2-x_2) - \frac{x_1x_2}{x_1}(x_1+x_2) = (x_1x_2-x_2) - x_2(x_1+x_2) = -x_2^2 - x_2$$

And the remainder

$$-x_2^2 - x_2 \xrightarrow{p_4} -x_2^2 - x_2 + (x_2^2 + x_2) = 0$$

Once again we have a zero remainder, and once again we don't have to alter L or G.

The next element we pick is $(1, 4)$.

$$\begin{aligned} S_{1,4} =& \frac{x_1x_2^2}{x_1x_2}(x_1x_2 + x_1) - \frac{x_1x_2^2}{x_2^2}(x_2^2 + x_2) = x_2(x_1x_2 + x_1) - x_1(x_2^2 + x_2) = \\ =& 0 \end{aligned}$$

and as the S-polynomial becomes zero, then so will the remainder, meaning we don't have to add anything here either.

The final element in L is $(3, 4)$.

$$S_{3,4} = \frac{x_1x_2^2}{x_1}(x_1 + x_2) - \frac{x_1x_2^2}{x_2^2}(x_2^2 + x_2) = x_2^2(x_1 + x_2) - x_1(x_2^2 - x_1x_2 = x_1x_2 - x_2^3$$

And the remainder becomes

$$\begin{aligned} x_1x_2 - x_2^3 \xrightarrow{p_1} & x_1x_2 - x_2^3 - (x_1x_2 + x_1) = \\ =& x_1 + x_2^3 \xrightarrow{p_3} x_1 + x_2^3 - (x_1 + x_2) = \\ =& x_2^3 - x_2 \xrightarrow{p_4} x_2^3 - x_2 - x_2(x_2^2 + x_2) = \\ =& x_2^2 + x_2 \xrightarrow{p_4} x_2^2 + x_2 - (x_2^2 + x_2) = 0 \end{aligned}$$

As L is now empty, the Buchberger algorithm is now complete, and our final basis is $\{x_1x_2 + x_1, x_1x_2 - x_2, x_1 + x_2, x_2^2 + x_2\}$

Note that there is a large degree of freedom when implementing this algorithm: apart from monomial ordering, we can pick the pairs of L in any order we like, and when reducing $S_{i,j}$ we can pick the reduction order of the polynomials. This can be of great help when trying to implement the algorithm as efficiently as possible. For example, in Example 14, we calculated the remainder of $S_{2,4}$ with respect to G. Here we had a choice in ordering our basis: if we had calculated $\text{rem}(S_{2,4}, (p_1, p_2, p_3, p_4))$, we would have gotten the reduction chain $x_1x_2 + x_2^2 \xrightarrow{p_1} x_2^2 - x_1 \xrightarrow{p_3} x_2^2 + x_2 \xrightarrow{p_4} 0$. We could also have calculated $\text{rem}(S_{2,4}, (p_2, p_3, p_4, p_1))$, and gotten the chain $x_1x_2 + x_2^2 \xrightarrow{p_2} x_2^2 + x_2 \xrightarrow{p_4} 0$, making the chain shorter. When dealing with larger cases, these types of reorderings can be immense time savers [3].

Also note that the result from the Buchberger algorithm is not unique, it actually depends on the various orderings we decide on. Finally, note that we never remove polynomials from G, we only add them. We would have to be very lucky if we were to end up with a reduced and minimal Gröbner basis. As we are not interested in reducing or minimizing our bases, however, we will manage perfectly with the Buchberger algorithm on its own.

3 Boolean formulas and $\mathbb{Z}_2$-polynomials

3.1 From formula to polynomial

We initially set out to find a way of deciding whether it is possible to satisfy a boolean formula. So far this problem seems largely untouched, as the tools we discussed in the previous section mainly related to abstract algebra and polynomial rings. In this section however, we will show how a boolean formula can be converted to a $\mathbb{Z}_2$-polynomial, while preserving satisfiability as solvability. After that we will hopefully find our algebraic tools more useful.

We will only discuss boolean formulas consisting of variables $(x_1, x_2, \ldots, x_n)$, disjunctions ("or"-signs, $\vee$), conjunctions ("and"-signs, $\wedge$) and negations ("not"-signs, $\neg$). This is not a large restriction since all other boolean signs, such as implication ($\Rightarrow$) can be rewritten using the allowed ones. $A \Rightarrow B$ is for example equivalent with $\neg A \vee B$.

First some notation: Given a boolean formula ϕ containing the boolean variables $\psi_1, \ldots, \psi_n$, a **variable assignment** is a function v that for each variable assigns a boolean value, $v : \{\psi_1, \ldots, \psi_n\} \to \{\top, \bot\}$. We can evaluate ϕ using v simply by replacing each occurrence of ψ_i with $v(\psi_i)$ in ϕ for each i. We write the evaluation of ϕ using v as $\phi(v)$. Note the similarity with how polynomials are evaluated.

Starting with a boolean formula ϕ, we can recursively transform it into a $\mathbb{Z}_2$-polynomial $T(\phi)$ using the following rules. Note here that φ denotes a boolean subformula.

- If ϕ consists of a single boolean variable, $\phi = \psi_i$, then let $T(\phi) = x_i$.
- If ϕ consists of a negation, $\phi = \neg\varphi$, then let $T(\phi) = T(\varphi) + 1$.
- If ϕ consists of a conjunction, $\phi = \varphi_1 \wedge \varphi_2$, then let $T(\phi) = T(\varphi_1) + T(\varphi_2) + T(\varphi_1)T(\varphi_2)$.
- If ϕ consists of a disjunction, $\phi = \varphi_1 \vee \varphi_2$, then let $T(\phi) = T(\varphi_1)T(\varphi_2)$.

Example 15. Let $\phi = (\psi_1 \wedge \neg\psi_2) \vee \neg\psi_1$. Then

$$T(\phi) = (x_1 + (x_2 + 1) + x_1(x_2 + 1))(x_1 + 1) = x_1 + x_2 + 1 + x_1^2 x_2$$

Note that the final equality is due to the fact that we are in $\mathbb{Z}_2$.

This transformation was given in [2], and is a modification of the Stone transformation [5].

It is clear that the process will end and that the rules will produce a proper polynomial in $\mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, where n is the number of variables in ϕ. We will now show that the rules also preserve satisfiability as solvability: let's assume we have a variable assignment v which satisfies the boolean formula ϕ. If we let the boolean value $\top$ correspond to the $\mathbb{Z}_2$-value 0 and $\bot$ correspond to 1, we can create a point $v' = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}_2^n$, so that the a_i correspond to $v(\psi_i)$.

Example 16. If $v(\psi_1) = \top, v(\psi_2) = \bot, v(\psi_3) = \bot$, then $v' = (0, 1, 1)$.

Theorem 3.1. *For an arbitrary boolean formula ϕ and variable assignment v, we have that v will satisfy ϕ if and only if v' solves $T(\phi)$. This means that the transformation perfectly preserves satisfiability as solvability.*

Proof. It is enough to show that $\phi(v) = \top \Leftrightarrow T(\phi)(v') = 0$, since this is equivalent to $\phi(v) = \bot \Leftrightarrow T(\phi)(v') = 1$.

We will now go through the recursion steps one by one, and, by an induction-like reasoning, show that each transformation preserves solutions.

- The base case is that ϕ consists of a single variable ψ_i: if $v(\psi_i) = \top$, then ϕ is satisfied. Due to the correspondence, the i:th coordinate in v' is then zero, and so v' solves the corresponding polynomial $T(\phi) = x_i$. The same reasoning can be used to show the other direction of the implication. Hence v satisfying ϕ corresponds precisely with v' solving $T(\phi)$.

- Negation: Assume that v satisfies φ precisely when v' solves the polynomial $T(\varphi)$ (so $\varphi(v) = \top \Leftrightarrow T(\varphi)(v') = 0$), and let $\phi = \neg\varphi$, so $T(\phi) = T(\varphi) + 1$.

  Assume that $T(\phi)(v') = 0$, so $T(\varphi)(v') = 1$. Since v' does not solve $T(\varphi)$, we have $\varphi(v) = \bot$ according to the induction assumption. This gives us that $\phi(v) = \neg\varphi(v) = \top$, and so ϕ is satisfied when $T(\phi)$ is solved.

  For proving the other direction of the equivalence, we do the same: Assuming that $\phi(v) = \top$ will similarly show us that $T(\phi)(v') = 0$. In total we have that if $\varphi(v) = \top$ precisely when $T(\varphi)(v') = 0$, then $\neg\varphi(v) = \top$ precisely when $T(\neg\varphi)(v') = 0$.

- Conjunction: assume that $\varphi_i(v) = \top \Leftrightarrow T(\varphi_i)(v') = 0$ for $i = 1, 2$ and let $\phi = \varphi_1 \wedge \varphi_2$. This means that $T(\phi) = T(\varphi_1) + T(\varphi_2) + T(\varphi_1)T(\varphi_2)$.

  Assume that $T(\phi)(v') = 0$. As we are in $\mathbb{Z}_2$, this is only possible if $T(\varphi_1)(v') = T(\varphi_2)(v') = 0$, since if either or both of them are 1, then $T(\phi)$ would be 1 as well. According to the induction assumption, we then have $\varphi_1(v) = \varphi_2(v) = \top$, implying that $\phi(v) = \varphi_1 \wedge \varphi_2 = \top$ as well. We have that $T(\varphi_1 \wedge \varphi_2)(v') = 0 \Rightarrow (\varphi_1 \wedge \varphi_2)(v) = \top$.

  Conversely, assume that $\phi(v) = \top$. This must mean that $\varphi_1(v) = \varphi_2(v) = \top$ too, and so according to the induction assumption $T(\varphi_1)(v') = T(\varphi_2)(v') = 0$. This in turn implies that $T(\phi)(v') = T(\varphi_1)(v') + T(\varphi_2)(v') + T(\varphi_1)(v')T(\varphi_2)(v') = 0 + 0 + 0 = 0$, so $\phi(v) = \top \Rightarrow T(\phi)(v') = 0$. In total, we have that if $\varphi_i(v) = \top \Leftrightarrow T(\varphi_i)(v') = 0$ for $i = 1, 2$, then $(\varphi_1 \wedge \varphi_2)(v) = \top \Leftrightarrow T(\varphi_1 \wedge \varphi_2)(v') = 0$

- Disjunction: once again assume that $\varphi_i(v) = \top \Leftrightarrow T(\varphi_i)(v') = 0$ for $i = 1, 2$, and let $\phi = \varphi_1 \vee \varphi_2$. Then $T(\phi)(v) = T(\varphi_1)(v')T(\varphi_2)(v') = 0$ implies either $T(\varphi_1)(v') = 0$, $T(\varphi_2)(v') = 0$, or both. According to the assumption, this implies either $\varphi_1(v) = \top$, $\varphi_2(v) = \top$ or both. In all three cases, we end up with $\phi(v) = \varphi_1(v) \vee \varphi_2(v) = \top$. We have that $T(\varphi_1 \vee \varphi_2)(v') = 0 \Rightarrow (\varphi_1 \vee \varphi_2)(v') = \top$.

  Conversely, assume that $\phi(v) = (\varphi_1 \wedge \varphi_2)(v) = \top$. We will by the same reasoning arrive at the conclusion that $\phi(v) = \top \Rightarrow T(\phi)(v') = 0$, and in total $\varphi_i(v) = \top \Leftrightarrow T(\varphi_i)(v') = 0$ for $i = 1, 2$ implies $T(\varphi_1 \vee \varphi_2)(v') = 0 \Leftrightarrow (\varphi_1 \vee \varphi_2)(v') = \top$

□

We have now shown that the transformation perfectly preserves satisfiability as solvability: If and only if there exists a variable assignment v so that $\phi(v) = \top$, then there is a point v' so that $T(\phi)(v') = 0$. We are finally ready to discuss our original problem:

Problem (SAT). *Given a boolean formula ϕ, is there any variable assignment v so that $\phi(v) = \top$? In other words, is ϕ satisfiable?*

We know that for every boolean formula ϕ, there exists a polynomial $T(\phi) \in \mathbb{Z}_2[x_1, x_2, \dots, x_n]$. We also know that if, and only if, ϕ is satisfiable, then there exists a point $v' \in \mathbb{Z}_2^n$ so that $T(\phi)(v') = 0$. The SAT-problem can hence be reformulated as:

Problem ($\mathbb{Z}_2$-polynomial solvability problem)**.** *Given a polynomial $p \in \mathbb{Z}_2[x_1, x_2, \dots, x_n]$, are there any points in $\mathbb{Z}_2^n$ which solves p?*

If we can answer the $\mathbb{Z}_2$-polynomial solvability problem, then we can easily answer the SAT-problem: Convert ϕ to the polynomial $T(\phi)$, and insert that into the $\mathbb{Z}_2$-polynomial solvability problem. The resulting answer will be the same as the answer to the SAT-problem.

We can reformulate the problem once more using ideals: let $I = \langle p \rangle$ be an ideal. We then know from Theorem 2.2 that $V_{\mathbb{Z}_2}(I)$ contains precisely the solutions to p with coordinates in $\mathbb{Z}_2$. If $V_{\mathbb{Z}_2}(I)$ is empty, then p is unsolvable, and ϕ is unsatisfiable. We have arrived at the next evolution of our problem:

Problem ($\mathbb{Z}_2$-variety problem)**.** *Given an ideal $I = \langle p \rangle, I \subseteq \mathbb{Z}_2[x_1, x_2, \dots, x_n]$, is $V_{\mathbb{Z}_2}(I)$ non-empty?*

We can generalize our results slightly - imagine that we had several boolean formulas $\varphi_1, \varphi_2, \dots, \varphi_m$, and we wanted to find out if it was possible to satisfy them all at once using the same variable assignment. This is equivalent to finding a solution to the equation system $T(\varphi_1) = T(\varphi_2) = \dots = T(\varphi_m) = 0$. Luckily, the ideal and variety approach still work. We can solve the problem of satisfying several boolean formulas by solving the generalized $\mathbb{Z}_2$-variety problem:

Problem (Generalized $\mathbb{Z}_2$-variety problem)**.** *Given an ideal $I = \langle p_1, p_2, \dots, p_m \rangle, I \subseteq \mathbb{Z}_2[x_1, x_2, \dots, x_n]$, is $V_{\mathbb{Z}_2}(I)$ non-empty?*

3.2 Field equations

Let $f = x^p - x \in \mathbb{Z}_p[x]$, where p is a prime. Is f solvable? We can easily answer this using Fermat's little theorem:

Theorem 3.2 (Fermat's little theorem)**.** *If p is a prime and a is an arbitrary integer, then $a^p \equiv a \bmod p$.*

Fermat's little theorem is well known and a proof can be found in, for example, [14].

Theorem 3.3. *Let $f = x^p - x \in \mathbb{Z}_p[x]$, where p is a prime. Then f is solved by the elements in $\mathbb{Z}_p$. No other solutions exist, even in the algebraic closure.*

Proof. Let a be an arbitrary element in $\mathbb{Z}_p$. According to Fermat's little theorem, $a^p - a \equiv 0 \bmod p$. This means that a solves f. Note that the degree of f is p, so f has at most p solutions. Since $\mathbb{Z}_p$ contains p elements, no other solutions can exist. $\square$

Our polynomial $x^p - x$ is a so called **field equation**, named so since its solutions are the elements of the field $\mathbb{Z}_p$. The situation can be generalized to several variables:

Definition 3.4 (Field equations)**.** *The* **field equations** *of a polynomial ring $\mathbb{Z}_p[x_1, x_2, \dots, x_n]$ are the polynomials*

$$x_1^p - x_1, x_2^p - x_2, \dots, x_n^p - x_n$$

Theorem 3.5. *Let $R = \mathbb{Z}_p[x_1, x_2, \dots, x_n]$. The field equations of R are solved by the points in $\mathbb{Z}_p^n$. No other solutions exists.*

Proof. Say that we have a point $a = (b_1, b_2, \dots, b_n)$ which solves the field equations. As a solves, for example, $x_1^p - x_1$, we must have that b_1 solves the polynomial f from Theorem 3.3, meaning that b_1 must be an element of $\mathbb{Z}_p$. As b_1 doesn't appear in the other equations, this is the only restriction. The same reasoning can be applied to the other field equations, eventually giving us that they are solved when all $b_i \in \mathbb{Z}_p$. In other words, the field equations are solved by, and only by, the elements of $\mathbb{Z}_p^n$. $\square$

Note that when we are in $\mathbb{Z}_2$, the field equations are $x_i^2 - x_i$, which can also be written as $x_i^2 + x_i$. In order to reduce clutter, we will use the latter version.

Instead of viewing the field equations as a system of equations, we can look at the ideal generated by them: $\langle x_1^p - x_1, x_2^p - x_2, \dots, x_n^p - x_n \rangle$.

Theorem 3.6. *Let $R = \mathbb{Z}_p[x_1, x_2, \dots, x_n]$ be a polynomial ring. The ideal $J = \langle x_1^p - x_1, x_2^p - x_2, \dots, x_n^p - x_n \rangle$, generated by the field equations of R, has the variety $V(J) = \mathbb{Z}_p^n$.*

Proof. Recall from Theorem 2.2 that the variety of an ideal consists of the solutions to its generators. Theorem 3.5 says that the solutions of the field equations are precisely the points in $\mathbb{Z}_p^n$. $\square$

These results may seem trivial, but turn out to be very useful: say that we have a set of polynomials $S \subseteq \mathbb{Z}_p[x_1, x_2, \dots, x_n]$, and that we are interested in finding out if the polynomials in S have any solutions with coordinates in $\mathbb{Z}_p$.

Since an ideal is solved by the solutions to its generators, one approach would be to let I be the ideal generated by the polynomials in S: $I = \langle p_1, \dots, p_m \rangle$. Then check if $V_{\mathbb{Z}_p}(I)$ was empty, that is, if the variety over $\mathbb{Z}_p^n$ was empty: if it is, there are no solutions with coordinates in $\mathbb{Z}_p$. But since $\mathbb{Z}_p$ is not algebraically closed, this is not easily done.

It would be far easier to find whether the entire variety $V(I)$ was empty: Hilbert's Nullstellensatz, Theorem 2.4 gives us one way, check if 1 is in the ideal or not. But that variety may contain points with coordinates in the algebraic closure, outside $\mathbb{Z}_p$, which we are not interested in.

However, the field equations give us a third option. Let $J = \langle x_1^p - x_1, x_2^p - x_2, \dots, x_n^p - x_n \rangle$ be the ideal generated by the field equations of $\mathbb{Z}_p[x_1, x_2, \dots, x_n]$. Let us examine $V(I + J)$.

Theorem 3.7. *Let I be an ideal in $\mathbb{Z}_p[x_1, x_2, \dots, x_n]$, and let J be the ideal generated by the field equations, $J = \langle x_1^p - x_1, x_2^p - x_2, \dots, x_n^p - x_n \rangle$. Then $V(I + J) = V_{\mathbb{Z}_p}(I)$.*

Proof. According to Theorem 2.3, $V(A + B) = V(A) \cap V(B)$ for arbitrary ideals A and B. This means that $V(I + J) = V(I) \cap V(J) = V(I) \cap \mathbb{Z}_p^n$, so $V(I + J)$ contains those points in $\mathbb{Z}_p^n$ which solve the polynomials in I. This means $V(I + J) = V_{\mathbb{Z}_p}(I)$. $\square$

We have that $V(I + J)$ contains the solutions of I which have coordinates in $\mathbb{Z}_p$, even though we consider the whole variety! By adding the field equations to the ideal, we can, so to speak, force the variety to only contain solutions with coordinates in $\mathbb{Z}_p$. Undoubtedly, this will prove useful.

3.3 Tying it all together

We are now ready to prove the following theorem:

Theorem 3.8 (Masterpiece Theorem)**.** *Let $I \subseteq \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$ be an arbitrary polynomial ideal, let J be the ideal generated by the field equations, $J = \langle x_1^p - x_1, x_2^p - x_2, \ldots, x_n^p - x_n \rangle$ and let G be a Gröbner basis of $I + J$. Then $V_{\mathbb{Z}_p}(I)$ will be empty if and only if $1 \in G$.*

Proof. Recall from Theorem 3.7 that $V_{\mathbb{Z}_p}(I) = V(I + J)$. According to Hilbert's Nullstellensatz, Theorem 2.4, $V(I + J)$ is empty if and only if $I + J = \mathbb{Z}_p[x_1, x_2, \ldots, x_n]$, or equivalently, if $1 \in I + J$. This, in turn, is equivalent to G containing 1, according to Theorem 2.11. □

We had previously reformulated the SAT-problem to this, seemingly more complicated, problem...

Problem. *Given an arbitrary ideal $I = \langle p_1, p_2, \ldots, p_m \rangle \subseteq \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, is $V_{\mathbb{Z}_2}(I)$ non-empty?*

Using the Masterpiece Theorem, we know this is equivalent to asking...

Problem. *Given an arbitrary polynomial ideal $I = \langle p_1, p_2, \ldots, p_m \rangle, I \subseteq \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, let J be the ideal generated by the field equations, $J = \langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n \rangle$, and let G be a Gröbner basis to $I + J$. Is $1 \in G$?*

We have finally reached a problem we can solve. Calculating a Gröbner basis G to $\langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n, p_1, p_2, \ldots, p_m \rangle$ using the Buchberger algorithm is easy, and we can return false immediately if the algorithm finds 1. If the algorithm stops without finding 1, we return true. This is our algebraic approach for solving the SAT-problem.

3.3.1 The Basic Algorithm

Let's recap the whole solution process: We started with the, seemingly simple, SAT-problem:

Given a boolean formula ϕ, is ϕ satisfiable?

And we solve this by...

Algorithm 4 The Basic Algorithm

```
Input: A boolean formula $\phi$
Output: True if $\phi$ is satisfiable, false otherwise
  Using the steps found in Section 3.1, transform $\phi$ to the polynomial $T(\phi) \in \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, where $n$ is the number of variables in $\phi$.
  Let $I = \langle x_1^2+x_1, x_2^2+x_2, \ldots, x_n^2+x_n, T(\phi) \rangle$. Using the Buchberger algorithm, calculate a Gröbner basis $G$ of $I$.
  if $1 \in G$ then
    return false
  else
    return true
  end if
```

If we have several boolean formulas $\varphi_1, \varphi_2, \ldots, \varphi_m$, let I be the ideal generated by all corresponding polynomials $T(\varphi_1), T(\varphi_2), \ldots, T(\varphi_m)$ and the field equations. In fact, in most cases this is the preferred approach - see Section 4.4 for more information.

Example 17. Say that we had a boolean formula $\phi = \psi_1 \wedge \neg\psi_2 \wedge \neg(\psi_1 \vee \neg\psi_2)$, and wanted to know if ϕ is satisfiable.

We begin by recursively transforming the formula to a polynomial:

$$\begin{aligned}\mathrm{T}(\phi) =&\mathrm{T}(\psi_1 \wedge \neg\psi_2 \wedge \neg(\psi_1 \vee \neg\psi_2)) = \\ =&\mathrm{T}(\psi_1 \wedge \neg\psi_2) + \mathrm{T}(\neg(\psi_1 \vee \neg\psi_2)) + \mathrm{T}(\psi_1 \wedge \neg\psi_2)\mathrm{T}(\neg(\psi_1 \vee \neg\psi_2))\end{aligned}$$

As intermediate steps, we calculate

$$\begin{aligned}\mathrm{T}(\psi_1 \wedge \neg\psi_2) =&\mathrm{T}(\psi_1) + \mathrm{T}(\neg\psi_2) + \mathrm{T}(\psi_1)\mathrm{T}(\neg\psi_2) = \\ =&x_1 + (\mathrm{T}(\psi_2) + 1) + x_1(\mathrm{T}(\psi_2) + 1) = x_1 + x_2 + 1 + x_1(x_2 + 1) = \\ =&x_1 + x_2 + 1 + x_1x_2 + x_1 = x_1x_2 + x_2 + 1\end{aligned}$$

We also calculate

$$\begin{aligned}\mathrm{T}(\neg(\psi_1 \vee \neg\psi_2)) =&1 + \mathrm{T}(\psi_1 \vee \neg\psi_2) = 1 + \mathrm{T}(\psi_1)\mathrm{T}(\neg\psi_2) = \\ =&1 + x_1(1 + \mathrm{T}(\psi_2)) = 1 + x_1(1 + x_2) = x_1x_2 + x_1 + 1\end{aligned}$$

Finally, we put them back in the original calculation:

$$\begin{aligned}\mathrm{T}(\phi) =&\mathrm{T}(\psi_1 \wedge \neg\psi_2) + \mathrm{T}(\neg(\psi_1 \vee \neg\psi_2)) + \mathrm{T}(\psi_1 \wedge \neg\psi_2)\mathrm{T}(\neg(\psi_1 \vee \neg\psi_2)) = \\ =&(x_1x_2 + x_2 + 1) + (x_1x_2 + x_1 + 1) + (x_1x_2 + x_2 + 1)(x_1x_2 + x_1 + 1) = \\ =&x_1x_2 + x_2 + 1 + x_1x_2 + x_1 + 1 + \\ &+ x_1^2x_2^2 + x_1^2x_2 + x_1x_2 + x_1x_2^2 + x_1x_2 + x_2 + x_1x_2 + x_1 + 1 = \\ =&x_1^2x_2^2 + x_1^2x_2 + x_1x_2^2 + x_1x_2 + 1\end{aligned}$$

So the polynomial corresponding to ϕ is $x_1^2x_2^2 + x_1^2x_2 + x_1x_2^2 + x_1x_2 + 1$. We call this polynomial p_3. If, and only if, ϕ is satisfiable, then there is a solution to p_3 in $\mathbb{Z}_2^n$.

The field equations corresponding to our ring are $p_1 = x_1^2+x_1$ and $p_2 = x_2^2+x_2$. Now we take a look at the ideal $I = \langle x_1^2+x_1, x_2^2+x_2, x_1^2x_2^2+x_1^2x_2+x_1x_2^2+x_1x_2+1\rangle \subseteq \mathbb{Z}_2[x_1, x_2]$. We initialize the Buchberger algorithm, by creating the sets $G = \{x_1^2+x_1, x_2^2+x_2, x_1^2x_2^2+ x_1^2x_2 + x_1x_2^2 + x_1x_2 + 1\}$, $L = \{(1, 2), (1, 3), (2, 3)\}$. We also decide to use deglex as our ordering. We choose the pair $(1, 3)$ at first, and calculate the S-polynomial...

$$\begin{aligned}S_{1,3} =&\frac{x_1^2x_2^2}{x_1^2}(x_1^2 + x_1) - \frac{x_1^2x_2^2}{x_1^2x_2^2}(x_1^2x_2^2 + x_1^2x_2 + x_1x_2^2 + x_1x_2 + 1) = \\ =&x_1^2x_2^2 + x_1x_2^2 + x_1^2x_2^2 + x_1^2x_2 + x_1x_2^2 + x_1x_2 + 1 = x_1^2x_2 + x_1x_2 + 1\end{aligned}$$

Calculating the remainder $\mathrm{rem}(S_{1,3}, (p_1, p_2, p_3))$ is the next step...

$$x_1^2x_2 + x_1x_2 + 1 \xrightarrow{p_1} x_1^2x_2 + x_1x_2 + 1 - x_2(x_1^2 + x_1) = 1$$

As this is a non-zero element, we add it to G. But then we can immediately conclude that 1 will be an element in the basis. This in turn means that the $V(I) = \emptyset$, and that ϕ is unsatisfiable.

Example 18. Let $\phi = \neg(\psi_1 \wedge \psi_2) \wedge (\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2))$

First we transform the formula into a polynomial...

$$\begin{aligned}\mathrm{T}(\phi) &= \mathrm{T}(\neg(\psi_1 \wedge \psi_2) \wedge (\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2))) \\ &= \mathrm{T}(\neg(\psi_1 \wedge \psi_2)) + \mathrm{T}(\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2)) + \mathrm{T}(\neg(\psi_1 \wedge \psi_2))\mathrm{T}(\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2))\end{aligned}$$

We calculate the intermediate expressions one at the time...

$$\begin{aligned}\mathrm{T}(\neg(\psi_1 \wedge \psi_2)) &= (1 + \mathrm{T}(\psi_1 \wedge \psi_2)) = (1 + (\mathrm{T}(\psi_1) + \mathrm{T}(\psi_2) + \mathrm{T}(\psi_1)\mathrm{T}(\psi_2) = \\ &= (1 + x_1 + x_2 + x_1x_2)\end{aligned}$$

$$\begin{aligned}\mathrm{T}(\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2)) &= \mathrm{T}(\psi_1)\mathrm{T}(\neg\psi_1 \wedge \neg\psi_2) = \\ &= x_1(\mathrm{T}(\neg\psi_1) + \mathrm{T}(\neg\psi_2) + \mathrm{T}(\neg\psi_1)\mathrm{T}(\neg\psi_2)) = \\ &= x_1((1 + \mathrm{T}(\psi_1)) + (1 + \mathrm{T}(\psi_2)) + (1 + \mathrm{T}(\psi_1))(1 + \mathrm{T}(\psi_2))) = \\ &= x_1((1 + x_1) + (1 + x_2) + (1 + x_1)(1 + x_2)) = \\ &= x_1(1 + x_1 + 1 + x_2 + 1 + x_1 + x_2 + x_1x_2) = \\ &= x_1(1 + x_1x_2) = x_1 + x_1^2x_2\end{aligned}$$

Putting these back into the original formula, we get

$$\begin{aligned}\mathrm{T}(\phi) =&\mathrm{T}(\neg(\psi_1 \wedge \psi_2)) + \mathrm{T}(\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2)) + \mathrm{T}(\neg(\psi_1 \wedge \psi_2))\mathrm{T}(\psi_1 \vee (\neg\psi_1 \wedge \neg\psi_2)) = \\ =&(1 + x_1 + x_2 + x_1x_2) + (x_1 + x_1^2x_2) + (1 + x_1 + x_2 + x_1x_2)(x_1 + x_1^2x_2) = \\ =&1 + x_1 + x_2 + x_1x_2 + x_1 + x_1^2x_2 + \\ &+ x_1 + x_1^2x_2 + x_1^2 + x_1^3x_2 + x_1x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^3x_2^2 = \\ =&1 + x_2 + x_1 + x_1^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^3x_2^2\end{aligned}$$

This is our corresponding polynomial. Now it's time for to calculate a Gröbner basis for the ideal generated by this polynomial and the field equations. We decide to use deglex as our monomial ordering, and define our generators as $p_1 = x_1^2 + x_1$, $p_2 = x_2^2 + x_2$, and let $p_3 = \mathrm{T}(\phi) = x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1$. Initially we have that $G = \{p_1, p_2, p_3\}$, and $L = \{(1, 2), (1, 3), (2, 3)\}$.

We decide to pick $(1, 2)$ from L first, and calculate the S-polynomial...

$$\begin{aligned}S_{1,2} &= \frac{\mathrm{lcm}(x_1^2, x_2^2)}{x_1^2}(x_1^2 + x_1) - \frac{\mathrm{lcm}(x_1^2, x_2^2)}{x_2^2}(x_2^2 + x_2) = \\ &= x_2^2(x_1^2 + x_1) - x_1^2(x_2^2 + x_2) = x_1^2x_2 + x_1x_2^2\end{aligned}$$

We then reduce the polynomial with the polynomials in G. Keep in mind that we can order the polynomials in G however we like.

$$\begin{aligned}&\mathrm{rem}(x_1^2x_2 - x_1x_2^2, (p_1, p_2, p_3)) : \\ &\quad x_1^2x_2 + x_1x_2^2 \xrightarrow{p_1} x_1^2x_2 + x_1x_2^2 - x_2(x_1^2 + x_1) = \\ &\quad =x_1x_2^2 + x_1x_2 \xrightarrow{p_2} x_1x_2^2 + x_1x_2 - x_1(x_2^2 + x_2) = 0\end{aligned}$$

The reduction ends as zero, so we do not alter G or L.

Next we pick the element $(1, 3)$ from L. The S-polynomial becomes...

$$\begin{aligned}S_{1,3} =&\frac{\mathrm{lcm}(x_1^2, x_1^3x_2^2)}{x_1^2}(x_1^2 + x_1) - \\ &\frac{\mathrm{lcm}(x_1^2, x_1^3x_2^2)}{x_1^3x_2^2}(x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1) = \\ =&x_1x_2^2(x_1^2 + x_1) - (x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1) = \\ =&x_1^3x_2 + x_1^2x_2 + x_1 + x_2 + 1\end{aligned}$$

...and the reduction gives us...

$$\begin{aligned}
&\text{rem}(x_1^3x_2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1, (p_1, p_2, p_3)) :\\
&\quad x_1^3x_2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1 \xrightarrow{p_1}\\
&\quad x_1^3x_2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1 - x_1x_2(x_1^2 + x_1) =\\
&\quad =x_1^2x_2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1 = x_1^2 + x_1 + x_2 + 1 \xrightarrow{p_1} x_2 + 1
\end{aligned}$$

As this polynomial is nonzero, we call it p_4 and add it to G, and add the pairs $(1, 4), (2, 4), (3, 4)$ to L. We currently have $G = \{x_1^2 + x_1, x_2^2 + x_2, x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1, x_2 + 1\}$, $L = \{(2, 3), (1, 4), (2, 4), (3, 4)\}$.

We pick another element from L, taking $(2, 3)$ this time...

$$\begin{aligned}
S_{2,3} =&\frac{\text{lcm}(x_2^2, x_1^3x_2^2)}{x_2^2}(x_2^2 + x_2)-\\
&\frac{\text{lcm}(x_2^2, x_1^3x_2^2)}{x_1^3x_2^2}(x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1) =\\
&=x_1^3(x_2^2 + x_2) - (x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1) =\\
&=x_1^2x_2^2 + x_1^2x_2 + x_1^2 + x_1 + x_2 + 1
\end{aligned}$$

The next step is to reduce the S-polynomial. Here we use the opportunity to pick the reduction order ourselves, and use (p_2, p_1, p_3, p_4) in order to gain a shorter chain.

$$\begin{aligned}
&\text{rem}(x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1, (p_2, p_1, p_3, p_4)) :\\
&\quad =x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1 \xrightarrow{p_2} x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1 - x_1^2(x_2^2 - x_2) =\\
&\quad =x_1 + x_2 + 1 \xrightarrow{p_4} 0
\end{aligned}$$

The polynomial is reduced to zero, so we do nothing with it. Instead we pick another element from L, choosing $(3, 4)$...

$$\begin{aligned}
S_{3,4} =& \frac{x_1^3x_2^2, x_1}{x_1^3x_2^2}(x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1) - \frac{x_1^3x_2^2, x_1}{x_1}(x_1 + x_2 + 1) =\\
&= (x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1) - x_1^2x_2^2(x_1 + x_2 + 1) =\\
&= x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1 - x_1^2x_2^3 - x_1^2x_2^2 =\\
&= x_1^3x_2 + x_1^2x_2 + x_1 + x_2 + 1 + x_1^2x_2^3
\end{aligned}$$

…and then we reduce it, in the order (p_1, p_4, p_2, p_3)…

$$\begin{aligned}
&\mathrm{rem}(x_1^2x_2^3 + x_1^3x_2 + x_1^2x_2 + x_1 + x_2 + 1, (p_1, p_4, p_2, p_3)) : \\
&\quad x_1^2x_2^3 + x_1^3x_2 + x_1^2x_2 + x_1 + x_2 + 1 \xrightarrow{p_1} \\
&\quad x_1^2x_2^3 + x_1^3x_2 + x_1^2x_2 + x_1 + x_2 + 1 - x_2^3(x_1^2 - x_1) = \\
&\; = x_1^3x_2 + x_1x_2^3 + x_1^2x_2 + x_1 + x_2 + 1 \xrightarrow{p_1} \\
&\quad x_1^3x_2 + x_1x_2^3 + x_1^2x_2 + x_1 + x_2 + 1 - x_1x_2(x_1^2 - x_1) = \\
&\; = x_1x_2^3 + x_1 + x_2 + 1 \xrightarrow{p_4} \\
&\quad x_1x_2^3 + x_1 + x_2 + 1 - x_2^3(x_1 + x_2 + 1) = \\
&\; = x_2^4 + x_2^3 + x_1 + x_2 + 1 \xrightarrow{p_4} \\
&\quad x_2^4 + x_2^3 + x_1 + x_2 + 1 - (x_1 + x_2 + x_1) = \\
&\; = x_2^4 + x_2^3 \xrightarrow{p_2} x_2^4 + x_2^3 + x_2^2(x_2^2 - x_2) = 0
\end{aligned}$$

Another reduction ending in zero, so, we don't have to do anything with this one.

The remaining elements in L are $(1, 4)$ and $(2, 4)$. We choose to take $(1, 4)$ first…

$$\begin{aligned}
S_{1,4} &= \frac{\mathrm{lcm}(x_1^2, x_1)}{x_1^2}(x_1^2 - x_1) - \frac{\mathrm{lcm}(x_1^2, x_1)}{x_1}(x_1 + x_2 + 1) = \\
&= (x_1^2 - x_1) - x_1(x_1 + x_2 + 1) = x_1x_2
\end{aligned}$$

…and reduce it…

$$\begin{aligned}
&\mathrm{rem}(x_1x_2, (p_1, p_2, p_3, p_4)) : \\
&\quad x_1x_2 \xrightarrow{p_4} x_1x_2 - x_2(x_1 + x_2 + 1) = \\
&\; = x_2^2 + x_2 \xrightarrow{p_2} 0
\end{aligned}$$

And finally, we pick the element $(2, 4)$ and get...

$$\begin{aligned}
S_{2,4} &= \frac{\mathrm{lcm}(x_2^2, x_1)}{x_2^2}(x_2^2 - x_2) - \frac{\mathrm{lcm}(x_2^2, x_1)}{x_1}(x_1 + x_2 + 1) = \\
&= x_1(x_2^2 - x_2) - x_2^2(x_1 + x_2 + 1) = x_2^3 + x_1x_2 + x_2^2
\end{aligned}$$

…and reduce it…

$$\begin{aligned}
&\mathrm{rem}(x_2^3 + x_1x_2 + x_2^2, (p_1, p_2, p_3, p_4)) : \\
&\quad x_2^3 + x_1x_2 + x_2^2 \xrightarrow{p_2} x_2^3 + x_1x_2 + x_2^2 - x_2(x_2^2 - x_2) = \\
&\quad x_1x_2 \xrightarrow{p_4} x_1x_2 - x_2(x_1 + x_2 + 1) = \\
&\; = x_2^2 + x_2 \xrightarrow{p_2} 0
\end{aligned}$$

The reduction once again ends in zero, so we don't have to do anything with this element as well. Now L is finally empty, so the Buchberger algorithm is finished. We ended up with the Gröbner basis $G = \{x_1^2 - x_1, x_2^2 - x_2, x_1^3x_2^2 + x_1^3x_2 + x_1^2x_2^2 + x_1^2x_2 + x_1 + x_2 + 1, x_1 + x_2 + 1\}$. Since $1 \notin G$, we can happily draw the conclusion that ϕ is, in fact, satisfiable.

We now have a working algorithm, but it is a very time consuming algorithm. Let's see if we can't improve it somewhat...

4 Technical discussion

We will now present some technical considerations for anyone interested in implementing this idea.

4.1 Internal Reduction

Remember that ideals can, in general, be generated in more than one way. In other words, if we have an ideal $I = \langle p_1, \ldots, p_k \rangle$, we may be able to find a different way of representing the same ideal, $I = \langle p'_1, \ldots, p'_l \rangle$. The speed of Buchberger algorithm obviously depends heavily on which generators we use to represent the ideal: finding a Gröbner basis for an ideal represented by many large polynomials will, in general, take longer time than finding a basis for an ideal represented by fewer smaller polynomials. For an arbitrary set of polynomials generating an ideal, is there an easy way of finding a simpler set generating the same ideal?

Yes there is. Say that we start with the ideal $I = \langle p_1, p_2 \rangle$, and that $\mathrm{lm}(p_2) | \mathrm{lm}(p_1)$. We can also, without loss of generality, assume that p_1, p_2 are monic. If we performed the Buchberger algorithm at once, we would first calculate $S_{1,2} = \frac{\mathrm{lt}(p_1)}{\mathrm{lt}(p_1)} p_1 - \frac{\mathrm{lt}(p_1)}{\mathrm{lt}(p_2)} p_2 = p_1 - \frac{\mathrm{lt}(p_1)}{\mathrm{lt}(p_2)} p_2$. But that polynomial is simply p_1 with the highest term reduced by p_2. The next step of the algorithm is to reduce $p_1 - \frac{\mathrm{lt}(p_1)}{\mathrm{lt}(p_2)} p_2$ further, but since it can not be reduced with p_1 (The leading term of p_1 has been removed already), our only choice is to reduce it more with p_2. The end result is simply $\mathrm{rem}(p_1, (p_2))$, which we denote p'_1, and if it is non-zero, we add it to our basis, $G = \{p_1, p_2, p'_1\}$.

But notice that $p_1 = p'_1 + qp_2$, for some polynomial q, meaning that p_1 is no longer necessary to generate our ideal. We can initially represent our ideal as either $\langle p_1, p_2 \rangle$ or as $\langle p'_1, p_2 \rangle$: both ways give the same ideal. The difference is that if we had chosen the first way, the next step would have been to immediately add p'_1 anyway. By replacing p_1 with p'_1 in our initial representation, we will save ourselves one polynomial, making our basis smaller.

If $\mathrm{lm}(p_2) \nmid \mathrm{lm}(p_1)$, then $\mathrm{rem}(p_1, (p_2))$ is not necessarily the reduced S-polynomial, but it can still be used to generate the ideal, so replacing p_1 with p'_1 would not have affected our representation in this case either. In fact, since reductions tend to make our polynomials "smaller"[3], reductions like these are usually still helpful. We can always safely, and most of the time usefully, attempt to reduce p_1 with the other polynomials in the basis.

In fact, if we had had started with several generators, $I = \langle p_1, p_2, \ldots, p_m \rangle$, then we could have started with reducing p_1 with all the other polynomials in the basis, yielding the new representation $I = \langle p'_1, p_2, \ldots, p_m \rangle$, where $p'_1 = \mathrm{rem}(p_1, (p_2, p_3, \ldots, p_m))$[4]. Having found this new representation, we can simplify it further, by reducing p_2 by $(p'_1, p_3, \ldots, p_m)$. We can continue this form of reductions all the way up to p_m, obtaining the representation $I = \langle p'_1, p'_2, \ldots, p'_m \rangle$. If at any point a polynomial is reduced to zero, then it can simply be removed from the representation.

And why stop there? If possible, restart the process by reducing p'_1 with the other p'_i, and so on. When no more reductions can be made, that is, when a representation $I = \langle p_1, p_2, \ldots, p_m \rangle$ has been reached such that $\mathrm{rem}(p_i, (p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m)) = p_i$ for all $1 \leq i \leq m$, then and only then do we stop. The ideal should be the same, but

[3]Fewer terms of smaller degree, though this depends on the monomial ordering.

[4]We can, of course, choose to order the polynomials $p_2, p_3, \ldots, p_m$ in any way we like

the representation will yield a lower number of elements in the final basis, making the algorithm significantly faster.

The process of reducing each $p_i \in G$ until no further reductions are possible is called **internal reduction**, and is incredibly useful. There are even ways of implementing the Buchberger algorithm by applying internal reduction to G after adding reduced S-polynomials to it, as described in [3], though we will not discuss that method in this paper.

4.2 Field equations in $\mathbb{Z}_2$

The large freedom given to us by the Buchberger algorithm and the fact that we are working with a $\mathbb{Z}_2$-polynomial ideal containing the field equations means we can take many shortcuts.

4.2.1 Ideal generation

Before we perform internal reduction, we can simplify the representation of our ideal in another way.

Lemma 4.1. *Let I be an arbitrary ideal in $\mathbb{Z}_2[x_1, x_2, \dots, x_n]$ containing the field equation $x_i^2 + x_i$, let f_1 be an arbitrary polynomial, $f_1 = m_1 + m_2 + \dots + m_l$, and let m_k be an arbitrary monomial in f_1 divisible by x_i, so $m_k = \tilde{m}_k x_i^\beta, \beta \geqslant 1, x_i \nmid \tilde{m}_k$ for some monomial $\tilde{m}_k$. That is, $f_1 = m_1 + m_2 + \dots + \tilde{m}_k x_i^\beta + \dots + m_l$.*

Let f_2 be the same polynomial as f_1, but with the exponent of x_i in m_k reduced to one, so $f_2 = m_1 + m_2 + \dots + \tilde{m}_k x_i + \dots + m_l$. Then $p_1 \in I \Leftrightarrow p_2 \in I$.

Proof. If $\beta = 1$, then the result is true. If $\beta \geqslant 2$, then since the field equation $x_i^2 + x_i$ is in I, we have that $\tilde{m}_k x_i^{\beta-2}(x_i^2 + x_i) + f_1 = m_1 + m_2 + \dots + \tilde{m}_k x_i^\beta + \tilde{m}_k x_i^{\beta-2+2} + \tilde{m}_k x_i^{\beta-2+1} + \dots + m_l = m_1 + m_2 + \dots + \tilde{m}_k x_i^{\beta-1} + \dots + m_l \in I$. We have managed to lower the exponent by one. The process can be repeated with the new polynomial until the exponent becomes one, so $f_2 \in I$.

The same reasoning backwards can be used to show $f_2 \in I \Rightarrow f_1 \in I$. □

Repeated application of the above lemma gives us the next theorem:

Theorem 4.2. *Let I be an arbitrary ideal in $\mathbb{Z}_2[x_1, x_2, \dots, x_n]$ containing the field equations, $x_1^2 + x_1, x_2^2 + x_2, \dots, x_n^2 + x_n \in I$, and let f_1 be an arbitrary polynomial, $f_1 = m_1 + m_2 + \dots + m_l$, where $m_i = x_{\alpha_{i,1}}^{\beta_{i,1}} x_{\alpha_{i,2}}^{\beta_{i,2}} \dots x_{\alpha_{i,k_i}}^{\beta_{i,k_i}}, \beta_{i,j} \geqslant 1$. Let the polynomial f_2 be the same polynomial with all exponents reduced to one, so $f_2 = \tilde{m}_1 + \tilde{m}_2 + \dots + \tilde{m}_l$, where $\tilde{m}_i = x_{\alpha_{i,1}} x_{\alpha_{i,2}} \dots x_{\alpha_{i,k_i}}$. Then $f_1 \in I \Leftrightarrow f_2 \in I$.*

Example 19. Say that we have an ideal $I \subseteq \mathbb{Z}_2[x_1, x_2, \dots, x_n]$ containing the field equations, and we knew that $x_1^2 x_2 + x_2^3 + x_3^4 + 1$ is in I. Then we can also say that $x_1 x_2 + x_2 + x_3 + 1$ is in I, as well as $x_1^{50} x_2^{50} + x_2^{50} + x_3^{50} + 1$.

Another more subtle example: say that $1 + x_1 + x_1^2 + x_1^3 + x_1^4$ is in I. Then we get that $1 + x_1 + x_1 + x_1 + x_1$ is in I, but since we are in $\mathbb{Z}_2$, the x_1-variables will cancel each other out. This means we can draw the conclusion that $1 \in I$.

In other words, in a $\mathbb{Z}_2$-polynomial ideal I containing the field equations, a monomial is, more or less, defined only by the variables it contains. The exponents are practically irrelevant. Instead of working with a polynomial p, we can work with the, for all intents and purposes, equivalent polynomial where all exponents have been set to one. This can be used to simplify, for example, ideal generation:

Theorem 4.3. *Let $I \subseteq \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$ be an ideal generated by m polynomials and the field equations, $I = \langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n, p_{n+1}, p_{n+2}, \ldots, p_{n+m}\rangle$. Let $\tilde{p}_i$ be the same polynomial as p_i, but with all exponents set to one. Then I is also generated by the polynomials $\tilde{p}_i$ and the field equations, $I = \langle x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n, \tilde{p}_{n+1}, \tilde{p}_{n+2}, \ldots, \tilde{p}_{n+m}\rangle$.*

This is a direct consequence of Theorem 4.2.

In the Basic Algorithm, we have precisely such an ideal $\langle x_1^2 - x_1, \ldots, x_n^2 - x_n, p_{n+1}, p_{n+2}, \ldots, p_{n+m}\rangle \subseteq \mathbb{Z}_2[x_1, \ldots, x_n]$. Simplifying the polynomials p_i not only gives us smaller, more manageable exponents, but since we are in $\mathbb{Z}_2$, two identical terms will cancel each other out. Since a monomial is now uniquely defined by the variables in it, we will find that such cancellations more often occur. Even better, sometimes we may find that two polynomials become the same, lowering the number of generators.

The equivalence goes beyond ideal generators, however...

4.2.2 Field equation reductions

A large part of the Buchberger algorithm is spent calculating $\text{rem}(S_{i,j}, (x_1^2 + x_1, x_2^2 + x_2, \ldots, x_n^2 + x_n, p_{n+1}, p_{n+2}, \ldots, p_{n+m}))$. Since we can perform the reduction with the polynomials in any order we like, we can decide to perform the reductions with the field equations first. What happens when we reduce a polynomial with a field equation?

Conveniently enough, it turns out that we can use the same technique as in Theorem 4.2 to speed up the reduction process.

Theorem 4.4. *Let $R = \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, let $p_i = x_i^2 + x_i$ be the field equation associated with the arbitrary variable x_i and let $f = m_1 + m_2 + \ldots + m_k$ be an arbitrary polynomial in $\mathbb{Z}_2[x_1, x_2, \ldots, x_n]$. For each monomial $m_j = x_1^{\beta_{j,1}} x_2^{\beta_{j,2}} \ldots x_i^{\beta_{j,i}} \ldots x_n^{\beta_{j,n}}$, $\beta_{j,l} \geq 0$, let $\tilde{m}_j$ be the same monomial, but with the exponent of x_i as zero if $\beta_{j,i}$ is zero, or one if $\beta_{j,i}$ is non-zero. Note that if $\beta_{j,i}$ is zero or one, $\tilde{m}_j = m_j$.*

Then $f = m_1 + m_2 + \ldots + m_k \xrightarrow{p_i} \tilde{m}_1 + \tilde{m}_2 + \ldots + \tilde{m}_k$. In other words, reducing a polynomial with the field equation $x_i^2 + x_i$ is equivalent to lowering all x_i-exponents to one if they are non-zero, or keeping them at zero if they already are zero.

Proof. The leading term in p_i is x_i^2, for all admissible orderings. We try to reduce f one monomial at the time. For each monomial m_j, we have three cases:

1. $x_i \nmid m_j$
2. $x_i \mid m_j$ but $x_i^2 \nmid m_j$
3. $x_i^2 \mid m_j$

In the first and second case, $\text{lm}(p_i) \nmid m_j$, so $\tilde{m}_j = m_j$ and $m_j \xrightarrow{p_i} \tilde{m}_j$.

In the third case, we have that $m_j/\text{lm}(p_i) = m_j/x_i^2 = x_1^{\beta_{j,1}} \ldots x_i^{\beta_{j,i}-2} \ldots x_n^{\beta_{j,n}}$, so the first reduction step becomes $x_1^{\beta_{j,1}} \ldots x_n^{\beta_{j,n}} - (x_i^2 - x_i) x_1^{\beta_{j,i}} \ldots x_i^{\beta_{j,i}-2} \ldots x_n^{\beta_{j,n}} = x_1^{\beta_{j,1}} \ldots x_i^{\beta_{j,i}-1} \ldots x_n^{\beta_{j,n}}$. The exponent has been reduced by one. Similarly, each further reduction step will reduce the exponent by one, until the exponent reaches one, and so m_j has been reduced to $\tilde{m}_j$.

Since each reduction made this way only affects the term m_j, the others terms will not be affected until it is their turn to be reduced. This way, eventually all monomials m_j will be reduced to their corresponding $\tilde{m}_j$. $\square$

Example 20. We have that $x_1x_2^3x_3^2 + x_1^2x_2 + x_2^2 + x_1x_2 + x_1x_3 \xrightarrow{x_2^2-x_2} x_1x_2x_3^2 + x_1^2x_2 + x_2 + x_1x_2 + x_1x_3$.

The process can be generalized to reduction with multiple field equations: let $p_{\alpha_i} = x_{\alpha_i}^2 + x_{\alpha_i}$. A reduction with p_{α_1} will lower exponents of x_{α_1} to one without touching the rest of the polynomial, a reduction with p_{α_2} will lower the exponents of x_{α_2} to one, and so on. In the end, all exponents on the variables x_{α_i} will be lowered to one.

Example 21. Continuing our previous example, reducing $x_1x_2^3x_3^2+x_1^2x_2+x_2^2+x_1x_2+x_1x_3$ with all field equations gives us $x_1x_2x_3 + x_1x_2 + x_2 + x_1x_2 + x_1x_3$. As we are in $\mathbb{Z}_2$, the x_1x_2 terms cancel each other out, leaving us with $x_1x_2x_3 + x_2 + x_1x_3$.

The results are very useful for speeding up the calculation of $\mathrm{rem}(S_{i,j}, (x_1^2 + x_1, x_2^2 + x_2, \dots, x_n^2+x_n, p_{n+1}, p_{n+2}, \dots, p_{n+m}))$. Instead of reducing $S_{i,j}$ with the field equations, let $\tilde{S}_{i,j}$ be the S-polynomial with all exponents lowered to one, and then only reduce with the polynomials $p_{n+1}, \dots, p_{n+m}$.

These types of polynomials in $\mathbb{Z}_2[x_1, x_2, \dots, x_n]$ with one as the only possible coefficient, are called **boolean polynomials** because of how they represent boolean formulas this way. They can also be defined as the elements in the quotient ring $\mathbb{Z}_2[x_1, \dots, x_n]/\langle x_1^2 + x_1, \dots, x_n^2 + x_n\rangle$.

The idea can in fact be taken even further...

4.2.3 Multiplication Modification

There is in fact never any need to bother with those irritating exponents anywhere in the Basic Algorithm. When implementing the algorithm, simply treat all exponents as one, all the time. Monomials can be uniquely defined by which variables they contain, so all monomials can be written as $x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}$, where the exponents α_i can either take the value 0 or 1. This allows us to to represent monomials with integers between 0 and $2^n - 1$ - view the integer as a number $c_nc_{n-1}\dots c_2c_1$ in base 2, where the c_i are either one or zero. If the variable x_i divides the monomial, then we let c_i be one. For example, the monomial 1 can be represented with zero (no variables divide it), the monomial x_1x_3 can be represented with 5, which in base 2 is 101. This will significantly save on memory and reduce runtime. Multiplying two monomials $m_1 = x_1^{\alpha_1}x_2^{\alpha_2}\dots x_n^{\alpha_n}, m_2 = x_1^{\beta_1}x_2^{\beta_2}\dots x_n^{\beta_n}$ simply becomes $m_1m_2 = x_1^{\max(\alpha_1,\beta_1)}\dots x_n^{\max(\alpha_n,\beta_n)}$. Using this form of multiplication, we perform multiplications and reduce the result with the field equations at the same time.

Example 22. Let $f_1 = x_1x_2 + x_1x_3 + x_2$ and let $f_2 = x_1x_2$. Then the product under our multiplication would be $f_1f_2 = x_1x_2 \cdot x_1x_2 + x_1x_2 \cdot x_1x_3 + x_1x_2 \cdot x_2 = x_1x_2 + x_1x_2x_3 + x_1x_2$, which is equivalent with $x_1x_2x_3$ as we are in $\mathbb{Z}_2$.

This type of multiplication is strongly recommended to be used from the start of the algorithm, in all steps: performing internal reduction, calculating S-polynomials, and reducing them. This completely eliminates the need of reducing with field equations, as those operations are handled automatically.

Note though that there is still a need to consider the field equations when adding elements to L: we have not eliminated the need for creating their S-polynomials. Also note that the multiplication must be used carefully when creating these S-polynomials - make sure that the field equation itself isn't canceled out. In Section 4.3 we will introduce a simplification which eliminates the need for manually calculating the S-polynomials involving field equations at all.

4.2.4 Order Preservation

The downside to this type of multiplication is that it does not preserve the monomial order. For example, using deglex and $x_1 \succ x_2$, we have that $x_1 + x_2$ is a correctly sorted polynomial, and multiplying it with x_1 gives us $x_1 \cdot (x_1 + x_2) = x_1^2 + x_1x_2$, which is still correctly sorted. Using our new multiplication, however, we get $x_1 \cdot (x_1 + x_2) = x_1 + x_1x_2$, where the correct sorting should be $x_1x_2 + x_1$.

This may not seem like such a big problem, but keeping the terms in the polynomial ordered is essential - we want to be able, at a glance, to find the leading term of any polynomial. We could store the polynomials in a data structure which kept track of the leading term, such as a heap. Another approach would be to resort the terms each time we needed them sorted, but those are both time-consuming and inefficient. A cleverer multiplication is probably in order: if we multiply a sorted polynomial $f = m_1 + m_2 + \ldots + m_k$ with a single variable x_i, we could split f into $f_1 + f_2$, where $f_1 = m_{\alpha_1} + m_{\alpha_2} + \ldots + m_{\alpha_{k_1}}$ consists of the terms divisible by x_i and $p_2 = m_{\beta_1} + m_{\beta_2} + \ldots + m_{\beta_{k_2}}$ consists of the terms which are not divisible by x_i. Using our new multiplication, we get that $x_i \cdot f_1 = f_1$, so this polynomial is already sorted. Similarly, we get that $x_i \cdot f_2$ is simply each monomial in f_2 multiplied by x_i, but since all monomials did not originally contain x_i, the order is still preserved.

We now have two correctly sorted polynomials f_1 and $x_i \cdot f_2$, and we wish to know their correctly sorted sum. This can be done with a technique called merging: we know that the leading term of $f_1 + x_i \cdot f_2$ is either the leading term of f_1 or the leading term of $x_i \cdot f_2$. So we simply compare the two, to find the one with highest order: if $\mathrm{lt}(f_1) \succ \mathrm{lt}(x_i \cdot f_2)$, then we let $\mathrm{lt}(f_1)$ be the leading term in $f_1 + x_i \cdot f_2$, otherwise, we let $\mathrm{lt}(x_i \cdot f_2)$ be the leading term.

Say that $\mathrm{lt}(f_1)$ was the leading term. What is then the next term in $f_1 + x_i \cdot f_2$? Well, that would have to be either $\mathrm{lt}(x_i \cdot f_2)$ or the next term in f_1, that is $\mathrm{lt}(\mathrm{t}(f_1))$. So we compare those two, and let the term of highest order be the second term in $f_1 + x_i \cdot f_2$.

Continue like this: compare the two leading terms of what remains of f_1 and $x_i \cdot f_2$, and add the one of highest order to $f_1 + x_i \cdot f_2$. If the two terms should happen to be equal, then remove both, as $1 + 1 = 0$ in $\mathbb{Z}_2$. When either f_1 or $x_i \cdot f_2$ runs out of terms, simply add all remaining terms from the other polynomial to $f_1 + x_i \cdot f_2$. The time it will take to sort $x_i \cdot f$ this way will be linear in the number of terms f contains.

This can be generalized to multiplication with monomials containing more than one variable: if we are to multiply $M = x_{\alpha_1} x_{\alpha_2} \ldots x_{\alpha_k}$ with the polynomial f, then start by multiplying f with x_{α_1}, using the split/merge technique. Then continue with the other variables in M. If M contains α_k variables and f contains k terms, then this multiplication will at worst take $O(\alpha_k k)$ operations. Building a heap or resorting the terms using using a fast comparison-based algorithm will at worst take $O(k \log(k))$ operations, so if the number of variables in M is lower than the logarithm of the number of terms in f, this split/merge technique is probably faster.

Example 23. In this example, we use lex as our ordering. Say that we wish to multiply x_2 with $f = x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2 + x_3$. Note that f is correctly sorted.

Algorithm 5 Merging two sorted boolean polynomials

Here we view $"f_1"$, $"f_2"$ and $"f_1+f_2"$ as ordered lists of terms, where terms can easily be added or removed via the operations $+$ and $-$. For example, $"f_1" \longleftarrow "f_1" + m_1$ means that we add the term m_1 to the end of the list $"f_1"$. The intended data structure for $"f_1"$, $"f_2"$ and $"f_1+f_2"$ is linear linked lists.

```
Input: Two non-zero correctly sorted boolean polynomials $"f_1"$, $"f_2"$.
Output: The boolean polynomial $"f_1 + f_2"$ will be the correctly sorted sum of $"f_1"$
  and $"f_2"$
  $"f_1 + f_2" \longleftarrow 0$(Initially, the polynomial is zero)
  $m_1 \longleftarrow \mathrm{lt}("f_1")$ (The two candidates for the leading term of $f_1 + f_2$)
  $m_2 \longleftarrow \mathrm{lt}("f_2")$
  while $"f_1"$ and $"f_2"$ are both non-zero do
    if $"m_1" \succ "m_2"$ then
      $"f_1 + f_2" \longleftarrow "f_1 + f_2" + m_1$ (Add $m_1$ to the end of $"f_1 + f_2"$)
      $"f_1" \longleftarrow "f_1" - m_1$ (Remove the term from $"f_1"$ ...)
      $m_1 \longleftarrow \mathrm{lt}("f_1")$(... and let $m_1$ be the candidate for the next term in the sum)
    else
      if $m_2 \succ m_1$ then
        $"f_1 + f_2" \longleftarrow "f_1 + f_2" + m_2$
        $"f_2" \longleftarrow "f_2" - m_2$
        $m_2 \longleftarrow \mathrm{lt}("f_2")$
      else
        if $m_1 = m_2$ then
          $"f_1" \longleftarrow "f_1" - m_1$
          $m_1 \longleftarrow \mathrm{lt}("f_1")$
          $"f_2" \longleftarrow "f_2" - m_2$
          $m_2 \longleftarrow \mathrm{lt}("f_2")$ (Two identical terms found, remove both)
        end if
      end if
    end if
  end while
  if $"f_1"$ is zero then
    $"f_1 + f_2" \longleftarrow "f_1 + f_2" + "f_2"$ (Add the remaining terms in $"f_2"$ to the sum)
  else
    if $"f_1"$ is zero then
      $"f_1 + f_2" \longleftarrow "f_1 + f_2" + "f_2"$ (Add the remaining terms in $"f_2"$ to the sum)
    end if
  end if
  return $"f_1 + f_2"$
```

First go through the terms of f and split them into f_1 and f_2, depending on if they contain x_2 or not: we get that $f_1 = x_1x_2x_3 + x_1x_2 + x_2$ and $f_2 = x_1x_3 + x_3$. Note how $f = f_1 + f_2$, and how f_1 and f_2 are correctly sorted.

Next we perform the multiplication $x_2 \cdot f = x_2 \cdot f_1 + x_2 \cdot f_2$. Since $x_2 \cdot f_1 = f_1$, we do not need to alter the terms of f_1, and $x_2 \cdot f_2 = x_1x_2x_3 + x_2x_3$. Since no terms in f_2 contain x_2, the multiplication is order-preserving.

Finally, we need to merge the terms of f_1 and $x_2 \cdot f_2$ together. We call the leading terms of each polynomial m_1 and m_2, and initially we have $m_1 = x_1x_2x_3$, $m_2 = x_1x_2x_3$. Since the two terms are equal, they will cancel each other out, so we do not add them to the polynomial $x_2 \cdot f$, but instead let m_1 and m_2 be the next terms in f_1 and f_2 respectively.

We then get that $m_1 = x_1x_2$ and $m_2 = x_2x_3$. Since $m_1 \succ m_2$, we add m_1 to $x_2 \cdot f$, and replace m_1 with the next term, x_2. So far, we have that $x_2 \cdot f = x_1x_2$.

We once again compare $m_1 = x_2$ with $m_2 = x_2x_3$. Since $m_2 \succ m_1$, we add m_2 to $x_2 \cdot f$, so now $x_2 \cdot f = x_1x_2 + x_2x_3$. We also try to replace m_2 with the next term in $x_2 \cdot f_2$, but that was the final term in $x_2 \cdot f_2$. This means that we can simply dump the remaining terms from f_1 to $x_2 \cdot f$ and call it a day: we get that $x_2 \cdot f = x_1x_2 + x_2x_3 + x_2$. The polynomial is, as expected, sorted.

4.3 Simpler S-polynomials

The calculation of an S-polynomial can in many cases be simplified, or even skipped entirely.

4.3.1 The Two Criteria

Theorem 4.5 (The Prime Criterion). *Let p_i, p_j be two polynomials in the ring $k[x_1, x_2, \ldots, x_n]$, and let $\mathrm{lm}(p_i)$ and $\mathrm{lm}(p_j)$ be relatively prime. Then $S_{i,j}$ will always be reduced to zero.*

Proof. This means that $\mathrm{lcm}(\mathrm{lt}(p_i), \mathrm{lt}(p_j))$ is simply $\mathrm{lt}(p_i)\mathrm{lt}(p_j)$. We calculate the S-polynomial and reduce with p_i, p_j.

$$\begin{aligned}
S_{i,j} =& \frac{\mathrm{lcm}(\mathrm{lt}(p_i), \mathrm{lt}(p_j))}{\mathrm{lt}(p_i)}(\mathrm{lt}(p_i) + \mathrm{t}(p_i)) - \frac{\mathrm{lcm}(\mathrm{lt}(p_i), \mathrm{lt}(p_j))}{\mathrm{lt}(p_j)}(\mathrm{lt}(p_j) + \mathrm{t}(p_j)) = \\
=& \mathrm{lt}(p_j)(\mathrm{lt}(p_i) + \mathrm{t}(p_i)) - \mathrm{lt}(p_i)(\mathrm{lt}(p_j) + \mathrm{lt}(p_j)) = \\
=& \mathrm{lt}(p_j)\mathrm{t}(p_i) - \mathrm{lt}(p_i)\mathrm{t}(p_j) \xrightarrow{p_j} \\
& \mathrm{lt}(p_j)\mathrm{t}(p_i) - \mathrm{lt}(p_i)\mathrm{t}(p_j) - \mathrm{t}(p_i)p_j = \\
=& -\mathrm{t}(p_j)\mathrm{t}(p_i) - \mathrm{lt}(p_i)\mathrm{t}(p_j) \xrightarrow{p_i} \\
& -\mathrm{t}(p_j)\mathrm{t}(p_i) + \mathrm{t}(p_i)\mathrm{t}(p_j) = 0
\end{aligned}$$

□

When picking a pair $(a, b) \in L$, it can be useful to check if p_a, p_b fulfill the condition of the prime criterion, thereby allowing us to discard the pair without performing any calculations, as we know the end result can be reduced to zero anyway.

Theorem 4.6 (The Chain Criterion). *If $\mathrm{lm}(p_k) | \mathrm{lcm}(\mathrm{lm}(p_i), \mathrm{lm}(p_j))$, and if $S_{i,k}$ and $S_{j,k}$ has already been calculated, then it is possible to reduce $S_{i,j}$ to zero. In other words, the pair (i, j) can be discarded in this case.*

The two criteria can be used in all implementations of the Buchberger algorithm, not just the special case described in this paper. For more information about the two criteria, as well as a proof of the chain criterion, please see [6].

Implementing checks for the chain criterion is slightly trickier, as it requires keeping track on which S-polynomials have already been calculated. The list L could concievably be used for this, if implemented with a different data structure. While using the chain criterion is highly recommended, we will not apply it in this paper, in order to keep the algorithm simple.

The criteria can also be used preemptively, as we will show in the next section.

4.3.2 The S-polynomials of field equations

The leading term of a field equation $p_i = x_i^2 + x_i$ is always x_i^2, regardless of monomial ordering. According to the prime criterion then, the S-polynomial $S_{i,j}$ of two field equations $p_i = x_i^2 + x_i$, $p_j = x_j^2 + x_j$, will always be reduced to zero. This means that we can skip checking all S-polynomials of two field equations.

But it doesn't end there: when p_i is the field equation $x_i^2 + x_i$ and p_j is an arbitrary polynomial such that $x_i \nmid \mathrm{lm}(p_j)$, the S-polynomial will also always be reduced to zero, allowing us to skip those pairs without calculating anything.

And in the case when $p_i = x_i^2 + x_i$ and $x_i | \mathrm{lm}(p_j)$, we can simply pre-calculate the result. We will assume that we have taken into account the exponent simplifications from Section 4.2, so that x_i only ever have the exponents zero and one. First we split up p_j into two smaller polynomials $x_i p_{j1}$ and p_{j2}, where $x_i p_{j1}$ contains the terms divisible by x_i and p_{j2} contains the terms not divisible by x_i. Note that no terms in either p_{j1} nor p_{j2} are divisible by x_i and $\mathrm{lt}(p_j) = x_i \mathrm{lt}(p_{j1})$. The S-polynomial then becomes

$$\begin{aligned}\frac{\mathrm{lcm}(x_i^2, \mathrm{lt}(p_j))}{x_i^2}(x_i^2 + x_i) - \frac{\mathrm{lcm}(x_i^2, \mathrm{lt}(p_j))}{\mathrm{lt}(p_j)} p_j =& \mathrm{lt}(p_{j1})x_i^2 + \mathrm{lt}(p_{j1})x_i + x_i p_j = \\ =& \mathrm{lt}(p_{j1})x_i^2 + \mathrm{lt}(p_{j1})x_i \\ &+ x_i(x_i \mathrm{lt}(p_{j1}) + x_i \mathrm{t}(p_{j1}) + p_{j2}) = \\ =& x_i \mathrm{lt}(p_{j1}) + x_i^2 \mathrm{t}(p_{j1}) + x_i p_{j2}\end{aligned}$$

This can be reduced with $x_i^2 - x_i$, which gives us...

$x_i \mathrm{lt}(p_{j1}) + x_i \mathrm{t}(p_{j1}) + x_i p_{j2} = x_i p_{j1} + x_i p_{j2}$

Now we reduce with $p_j = x_i p_{j1} + p_{j2}$, giving the end result $(x_i + 1)p_{j2}$. The polynomial is non-zero, and after reduction with the other polynomials in G, we might be required to add it to G.

We can now handle all cases of S-polynomials involving field equations:

Theorem 4.7. *Let p_i be the field equation $x_i^2 + x_i$.*

- *If p_j is another field equation, then $S_{i,j}$ can be reduced to zero and does not need to be calculated.*
- *If $x_i \nmid \mathrm{lm}(p_j)$, then $S_{i,j}$ can be reduced to zero and does not need to be calculated.*
- *If $x_i \mid \mathrm{lm}(p_j)$, then $S_{i,j}$ becomes $(x_i + 1)p_{j2}$, where p_{j2} consists of the terms in p_j not divisible by x_i. Note that we must still perform further reductions before we are done.*

4.4 Conjunctive normal form or not?

According to the original Basic Algorithm, we are to examine the ideal $I = \langle p_1, p_2, \dots, p_n, f \rangle$, where the p_i are field equations. The problem with this approach is that as the original boolean formula grows larger, so does f, until it becomes unmanageably large. Is there any way we can modify our algorithm in order to combat this problem?

Yes there is. Remember that according to the Masterpiece Theorem (Theorem 3.8) our approach works equally well when we are dealing with multiple boolean formulas, and so is working with the ideal $I = \langle p_1, p_2, \dots, p_n, f_1, \dots, f_m \rangle$. If we could split up our original boolean formula ϕ into smaller boolean formulas φ_i, we would instead deal with an ideal generated by more and smaller polynomials, which might be more desirable. Is this possible?

Luckily, all boolean formulas ϕ can be rewritten to a so called conjunctive normal form, or CNF. A CNF-formula is always on the form $\phi = \varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_k$, where each φ_i is a boolean formula written on the form $\varphi_i = (\psi_1 \vee \psi_2 \vee \dots \vee \psi_l)$, where the ψ_i are **literals**, that is, either variables or negations of variables. In other words, a CNF-formula is a conjunction of disjunctions of literals. In a CNF-formula, each smaller formula φ_i is called a **clause**.

Example 24. For example, the boolean formula $\phi = (\psi_1 \vee \neg\psi_2 \vee \psi_3) \wedge (\psi_2 \vee \neg\psi_3) \wedge (\neg\psi_1)$ is a proper CNF-formula.

Satisfying a CNF-formula $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_k$ is the same as satisfying each φ_i individually, using the same variable assignment. Instead of transforming the whole formula ϕ to a single polynomial $T(\phi)$ and examine the ideal $\langle p_1, \dots, p_n, T(\phi) \rangle$, we can first transform ϕ to a CNF-formula $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_k$, transform each φ_i to a polynomial $T(\varphi_i)$ and examine the ideal $\langle p_1, \dots, p_n, T(\varphi_1), T(\varphi_2), \dots, T(\varphi_k) \rangle$. This will give more generators, but smaller polynomials.

4.4.1 CNF-transformation

Transforming a boolean formula to a CNF-formula can be done in several ways. Some consist of simply rewriting the formula while preserving equivalence, but these might exponentially increase the size of the formula, leading to much larger polynomials. As we are only interested in preserving satisfiability, we are not afraid of introducing more variables, if it helps limit the number, and size, of clauses[5].

One such way transforming a regular boolean formula to a CNF-formula while limiting the size of the formula to linear growth is as follows: Let ϕ be a boolean formula, for example $\neg((\psi_1 \wedge \psi_2) \vee \psi_3)$. This formula consists of a number of subformulas: in our case the subformulas $\varphi_1 = (\psi_1 \wedge \psi_2)$, $\varphi_2 = (\varphi_1 \vee \psi_3)$ and $\varphi_3 = \neg\varphi_2$. For each inner subformula φ_i, create a new variable ϱ_i, representing the boolean value of the subformula.

Now we can create additional clauses for each subformula φ_i, so that ϱ_i is true if and only if φ_i is true: in our example, the clauses corresponding to the subformula φ_1 are $(\varrho_1 \vee \neg\psi_1 \vee \neg\psi_2) \wedge (\neg\varrho_1 \vee \psi_1) \wedge (\neg\varrho_1 \vee \psi_2)$. Here, the first clause means ϱ_1 can not be false while both ψ_1, ψ_2 are true, and the other two clauses mean ϱ_1 can not be true while either ψ_1 or ψ_2 are false. In order for all clauses to be satisfied, we must have that ϱ_1 is true precisely when both ψ_1 and ψ_2 are true, and false otherwise.

[5]More variables will mean more field equations, but if we take into account our earlier changes, many of the corresponding S-polynomials will be discarded, and the reductions will be done automatically.

Similarly, the clauses corresponding to the disjunction-formula φ_2 can be written as $(\neg\varrho_2 \vee \psi_3 \vee \varrho_1) \wedge (\varrho_2 \vee \neg\varrho_1) \wedge (\varrho_2 \vee \neg\psi_3)$. The final formula φ_3 can be written as the clauses $(\varrho_3 \vee \varrho_2) \wedge (\neg\varrho_3 \vee \neg\varrho_2)$. Putting all clauses corresponding to all subformulas φ_i together creates one large CNF-formula, which is satisfiable if and only if the original formula is satisfiable. For more detail in transforming boolean formulas this way, please check [16].

4.4.2 Does it help?

This alternate way of describing boolean formulas helps immensely: a rudimentary experiment in the open-source software SINGULAR, [7] have confirmed our suspicions that, in general, it is better to deal with several smaller polynomials rather than one large.

The test started with defining the field equations $p_1, \ldots, p_n \in \mathbb{Z}_2[x_1, x_2, \ldots, x_n]$, and then randomizing m polynomials p_i, $i = n+1, \ldots, n+m$. Then we let the program calculate the reduced Gröbner basis of the ideal $I = \langle p_1, \ldots, p_n, p_{n+1}, p_{n+2}, \ldots, p_{n+m}\rangle$. This case corresponds with rewriting our boolean formula and dealing with multiple smaller polynomials.

Afterwards, we defined the polynomial $f = (p_{n+1} - 1)(p_{n+2} - 1)\ldots(p_{n+m} - 1) - 1$, and note that f is solved precisely when the p_{n+i} are all solved. We then let the program calculate the reduced Gröbner basis of the ideal $J = \langle p_1, p_2, \ldots, p_n, f\rangle$. This case would correspond to transforming the boolean formula directly, and dealing with one large polynomial.

The results were very clear: Already when with 18 variables and ten random polynomials, there was a large difference between the two cases: the multipolynomial approach took 9 seconds, the single-polynomial approach took 611 seconds.

The experiment may be considered primitive, as well as not completely applicable to our situation; We would like to compare whether using CNF rather than an "untouched" formula would result in smaller times. Using CNF would result in more and smaller polynomials, as well as more variables and field equations. In our experiment, the two cases had the same number of variables and field equations, skewing the result in favor of the multi-polynomial case. Also, in the experiment, the large polynomial f was created by putting the smaller polynomials together, something that will not happen if f was based on an untouched boolean formula, making f much larger than necessary. This too skews the results towards the multi-polynomial case. Nevertheless, the results were so heavily in favor of the multi-polynomial case, that there is no doubt that this approach is more effective in practice.

4.4.3 An explanation: Why not the regular Stone transformation?

Eagle-eyed readers will wonder why on earth we used a modified Stone transformation back in Section 3.1, and we are now ready to answer that question.

The original Stone transformation transforms boolean expressions to polynomials in $\mathbb{Z}_2$, just like the transformation presnted in this paper, with the difference that the original Stone transformation represents $\top$ with 1 and $\perp$ with 0. This also means that the transformation rules are different: the original transformation lets $\wedge$ be represented by multiplication $(\mathrm{T}(\phi_1 \wedge \phi_2) = \mathrm{T}(\phi_1)T(\phi_2)$, and $\vee$ is represented by repeated addition $(\mathrm{T}(\phi_1 \vee \phi_2) = \mathrm{T}(\phi_1) + \mathrm{T}(\phi_2) + \mathrm{T}(\phi_1)\mathrm{T}(\phi_2))$

This representation may seem more intuitive at first, but anyone actually interested in using this method for solving the SAT-problem in real life, will undoubtedly first rewrite the formula to CNF, then treating each clause as a separate formula, as discussed above. This means that the formulas to be transformed will contain many disjunctions, and no conjunctions, meaning that we want our disjunctions to have as simple form as possible. For example, with our transformation, we get that

$$\begin{aligned} T(\psi_1 \vee \neg\psi_2 \vee \psi_3) =& x_1(1+x_2)x_3 = \\ =& x_1x_3 + x_1x_2x_3 \end{aligned}$$

whereas the original Stone transformation would have given us

$$\begin{aligned} T(\psi_1 \vee \neg\psi_2 \vee \psi_3) =& (x_1 + (1+x_2) + x_1(1+x_2)) + x_3 + (x_1 + (1+x_2) + x_1(1+x_2))x_3 = \\ =& 1 + x_2 + x_1x_2 + x_2x_3 + x_1x_2x_3 \end{aligned}$$

after simplification. As smaller polynomials are more manageable, a reasonable guess is that the modified transformation is more useful for our purposes.

4.4.4 S-polynomials and CNF-formulas

Another upside to CNF-representation with the modified Stone-transformation is that we get some more S-polynomials which can be discarded.

Theorem 4.8. *Let p_i be the field equation corresponding to the variable x_i, and let p_j be on the form $(x_i+c)p'$ for some constant c and polynomial p', where $\mathrm{lt}(p')$ is not divisible by x_i. Then $S_{i,j}$ can always be reduced to zero.*

Proof. We have that $\mathrm{lt}(p_j) = x_i\mathrm{lt}(p')$, and two cases: either $c = 0$ or $c = 1$. If $c = 0$, we get

$$\begin{aligned} S_{i,j} =& \frac{\mathrm{lcm}(x_i\mathrm{lt}(p'), x_i^2)}{x_i\mathrm{lt}(p')} x_ip' - \frac{\mathrm{lcm}(x_i\mathrm{lt}(p'), x_i^2)}{x_i^2}(x_i^2 - x_i) = \\ =& x_i^2(\mathrm{lt}(p') + \mathrm{t}(p')) - \mathrm{lt}(p')(x_i^2 - x_i) = \\ =& x_i^2\mathrm{t}(p') + x_i\mathrm{lt}(p') \xrightarrow{p_i} x_i\mathrm{t}(p') + x_i\mathrm{lt}(p') = x_ip' \xrightarrow{p_j} 0 \end{aligned}$$

In the second case, we have

$$\begin{aligned} S_{i,j} =& \frac{\mathrm{lcm}(x_i\mathrm{lt}(p'), x_i^2)}{x_i\mathrm{lt}(p')}(x_i+1)p' - \frac{\mathrm{lcm}(x_i\mathrm{lt}(p'), x_i^2)}{x_i^2}(x_i^2 - x_i) = \\ =& x_i^2(\mathrm{lt}(p') + \mathrm{t}(p')) + x_ip' + x_i^2\mathrm{lt}(p') + x_i\mathrm{lt}(p') = \\ =& x_i^2\mathrm{t}(p') + x_ip' + x_i\mathrm{lt}(p') \xrightarrow{p_i} \\ & x_i\mathrm{t}(p') + x_ip' + x_i\mathrm{lt}(p') = x_ip' + x_ip' = 0 \end{aligned}$$

In both cases, we end up with a zero. □

Our initial polynomials, the ones corresponding to boolean formulas, will all be containing factors $(x_i + c)$ for constants c, and so we could use this discard these S-polynomials.

However, we will not use this. As many polynomials corresponding to boolean formulas will be broken apart by the internal reduction, we will not probably not start with such conveniently factorizable polynomials. Might it be useful to apply the theorem to S-polynomials of field equations and arbitrary polynomials, by first checking if the arbitrary polynomial can be factorized in this way? Probably not - checking factorization is not easy, and using Theorem 4.3 is better in this case.

4.4.5 Interesting experiment details

Recall the experiment described in Section 4.4.2. An observation about the experiment is the following.

Theorem 4.9. *Let p_i, $i = 1, \dots, n$ be the set of field equations in $\mathbb{Z}_p[x_1, x_2, \dots, x_n]$, and let p_{n+i},$i = 1 \dots, m$ be a set of arbitrary polynomials in $\mathbb{Z}_p[x_1, x_2, \dots, x_n]$. Let $I = \langle p_1, \dots, p_n, p_{n+1}, \dots, p_{n+m} \rangle$ and $J = \langle p_1, p_2, \dots, p_n, f \rangle$, where $f = (p_{n+1} - 1)(p_{n+2} - 1) \dots (p_{m+n} - 1) - 1$. Then $I = J$!*

Even though these results may not be relevant for our algorithm, they are still interesting in their own right and deserve to be discussed. Impatient readers who do not appreciate the beauty of mathematics may wish to skip this section and watch some reality tv instead.

In order to prove this, we will need the following theorem:

Theorem 4.10. *Let $I, J \subseteq K[x_1, x_2, \dots, x_n]$ be two arbitrary radical ideals, and let K be algebraically closed. If $V(I) = V(J)$, then $I = J$.*

The proof is non-trivial, and can be found in [1].

We need to prove that our ideals are radical if we are to use this theorem. For the uninitiated, an ideal I is radical, if for each $r^k \in I$ we have $r \in I$, that is, if I contains all "roots" of its members.

Theorem 4.11. *An ideal $L \subseteq \mathbb{Z}_p[x_1, x_2, \dots, x_n]$ that contains the field equations is radical.*

Proof. Say that $r^k \in L$ for some polynomial $r = a_1m_1 + a_2m_2 + \dots + a_lm_l$, where the a_i are coefficients and m_i are monomials. Let d be the smallest exponentiation of p larger than k, so that $d = p^c > k$, and $p^{c-1} \leq k$ for some natural number c. We can then multiply r^k with r^{d-k} and get that $r^d \in L$.

$$r^d = (a_1m_1 + a_2m_2 + \dots + a_lm_l)^d = (a_1m_1 + (a_2m_2 + \dots + a_lm_l))^d =$$
$$= a_1^d m_1^d + (a_2m_2 + \dots + a_lm_l)^d + \sum_{i=1}^{d-1} \left(\frac{d!}{i!(d-i)!} \right) a_1^i m_1^i (a_2m_2 + \dots + a_lm_l)^{d-i}$$

Note that each term apart from the first two are divisible by d, and so is a multiple of p. As we are in $\mathbb{Z}_p$, they are canceled out. If we expand the second term similarly, we will eventually get

$$r^d = a_1^d m_1^d + a_2^d m_2^d + \dots + a_l^d m_l^d$$

The coefficients are equivalent to $a_i^d = a_i^{p^c}$. As $a_i^p = a_i$ since we are in $\mathbb{Z}_p$, we get that $a_i^{p^c} = a_i^{p^{c-1}} = \dots = a_i^p = a_i$. The monomials can be reduced with the field equations to m_i itself (remember, d is a multiple of p). In total, we have that $r^d = r$, so $r^k \in I \Rightarrow r \in L$, so the inclusion of field equations makes an ideal in $\mathbb{Z}_p$ radical. □

We are ready for the proof.

Proof of Theorem 4.9. We will use Theorem 4.10 for our proof.

First we need to show that our two ideals I and J have the same variety. For this we refer back to Section 4.4.2. Note that the variety is the same, no matter what the surrounding ring is.

If we were to use Theorem 4.10 directly, we would need that I and J were ideals in a ring with coefficients in an algebraically closed field. $\mathbb{Z}_p$ is not algebraically closed, but we can easily get over that little problem: consider the ideals I' and J' as the ideals with the same generators as I and J, but with the surrounding ring $\overline{\mathbb{Z}_2}$ instead. We have that $V(I') = V(J')$, that they are radical and have coefficients in an algebraically closed field, so I' and J' are the same ideal, according to Theorem 4.10.

But don't be fooled; I' is still generated by the same generators as I, all of which has coefficients in $\mathbb{Z}_2$. Say that we were to perform the Buchberger algorithm on I': then we would get a Gröbner basis $G_{I'}$. As $I' = J'$, this is also a basis for J'. Since the Buchberger algorithm doesn't take the surrounding coefficient field into account when creating $G_{I'}$, the Gröbner basis for I must be the same as the Gröbner basis for I', meaning that $G_{I'}$ is a Gröbner basis for I as well. Similarly, we can calculate a Gröbner basis $G_{J'}$ for J', which also generates J and I'.

Now take an arbitrary element $p \in I$. The normal form of p with respect to I is zero, so $\mathrm{NF}(p, G_{I'}) = 0$. But as $G_{I'}$ generates J' as well, the normal form of p with respect to J' is also zero, and so $\mathrm{NF}(p, G_{J'}) = 0$. But the normal form calculations does not take the surrounding ring into account either, so $\mathrm{NF}(p, G_{J'}) = 0$ implies that $p \in J$ as well. This means that the ideals are identical, even when the surrounding ring is not algebraically closed. $\square$

4.5 The Main Algorithm

Algorithm 5 is a recap of the whole algorithm, including the improvements from this section.

Given a boolean formula ϕ, is ϕ satisfiable? The algorithm can be performed in any term ordering, but it is strongly recommended that degrevlex is used. One unused freedom we have is that we can select the pair (a, b) however we like. There are clever methods of deciding which pairs to treat first (for example, to maximize the potential of the chain criterion), but we will not discuss this in detail.

Example 25. We will use degrevlex in this example. Let $\phi = (\neg\psi_1 \vee \psi_2 \vee \neg\psi_3) \wedge (\neg\psi_2 \vee \neg\psi_3) \wedge (\psi_1 \vee \neg\psi_2) \wedge (\psi_1 \vee \psi_4)$. Note that ϕ is a CNF-formula. We define p_1, p_2, p_3, p_4 as our field equations, and transform each clause from ϕ into a polynomial:

$$p_5 = (x_1+1)x_2(x_3+1) = x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2$$
$$p_6 = (x_2+1)(x_3+1) = x_2x_3 + x_2 + x_3 + 1$$
$$p_7 = x_1(x_2+1) = x_1x_2 + x_1$$
$$p_8 = x_1x_4$$

We set $G = \{x_1^2 + x_1, x_2^2 + x_2, x_3^2 + x_3, x_4^2 + x_4, x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2, x_2x_3 + x_2 + x_3 + 1, x_1x_2 + x_1, x_1x_4\}$ and perform internal reduction: One possible reduction at this

Algorithm 6 The Main Algorithm, part 1

```
Input: A boolean formula $\phi$
Output: True if $\phi$ is satisfiable, false otherwise.
  Rewrite $\phi$ to conjunctive normal form, $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_m$
  $n \leftarrow$ the number of variables in $\varphi_1 \wedge \ldots \wedge \varphi_m$
  for $i = 1$ to $n$ do
    Let $p_i = x_i^2 - x_i$
  end for
  for $i = 1$ to $m$ do
    Let $p_{n+i} = \mathrm{T}(\varphi_i)$, the transformed polynomial of the i'th clause. Remember that
    the transformation is done using the modified multiplication. The resulting poly-
    nomials will then only contain variables with the exponent 1.
  end for
  Let $G$ be the set of all $p_i$
  Perform internal reduction on $G$
  $n' \leftarrow= n + m$ ($n'$ is the number of elements in $G$)
  Let L be an empty set
  for $i = 1$ to $n$ do
    for $j = n + 1$ to $n'$ do
      Add $(i, j)$ to $L$ (Note that we do not need to consider S-polynomials of two field
      equations)
    end for
  end for
  for $i = n + 1$ to $n' - 1$ do
    for $j = i + 1$ to $n'$ do
      Add $(i, j)$ to $L$
    end for
  end for
```

Algorithm 7 The Main Algorithm, part 2

```
while $L$ is not empty do
  Select an element $(a, b)$ from $L$, and remove it.
  if $a < n$ (That is, if $p_a$ is a field equation representing a variable $x_a$) then
    if $x_a \mid \mathrm{lt}(p_b)$ then
      Let $S_{a,b} = (x_a + 1)\hat{p}$, where $\hat{p}$ are the terms in $p_b$ not divisible by $x_a$. Use the
      modified multiplication.
    else
      Restart the while-loop (The S-polynomial can be discarded, Theorem 4.3)
    end if
  else
    if $\gcd(\mathrm{lm}(p_a), \mathrm{lm}(p_b)) = 1$ (The prime criteria, Theorem 4.5) then
      Restart the while-loop
    else
      Calculate $S_{a,b}$, using the modified multiplication.
    end if
  end if
  Calculate $R = \mathrm{rem}(S_{a,b}, (p_{n+1}, \ldots, p_{n'})$, where the $p_i$ are ordered arbitrarily. Use
  the modified multiplication.
  if $R = 1$ then
    return false
  else
    if $R \neq 0$ then
      $n' \leftarrow n' + 1$
      $p_{n'} = R$
      Add $p_{n'}$ to $G$
      for $i = 1$to $n' - 1$ do
        Add $(i, n')$ to $L$
      end for
    end if
  end if
end while
return true
```

stage is to reduce p_5 with p_6...

$$x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2 \xrightarrow{p_6} x_1x_2x_3 + x_1x_2 + x_2x_3 + x_2 - x_1(x_2x_3 + x_2 + x_3 + 1) =$$
$$= x_1x_3 + x_2x_3 + x_2 + x_1 \xrightarrow{p_6} x_1x_3 + x_2x_3 + x_2 + x_1 - (x_2x_3 + x_2 + x_3 + 1) =$$
$$= x_1x_3 + x_1 + x_3 + 1$$

Now $p_5 = x_1x_3 + x_1 + x_3 + 1$, $G = \{x_1^2 + x_1, x_2^2 + x_2, x_3^2 + x_3, x_4^2 + x_4x_1x_3 + x_1 + x_3 + 1, x_2x_3 + x_2 + x_3 + 1, x_1x_2 + x_1, x_1x_4\}$, and since no leading terms divides any term in the other polynomials, no more reductions are possible, and the internal reduction is finished.

We then initiate the list

$$L = \{(1,5),(1,6),(1,7),(1,8),(2,5),(2,6),(2,7),(2,8),(3,5),(3,6),(3,7),(3,8),(4,5),$$
$$(4,6),(4,7),(4,8),(5,6),(5,7),(5,8),(6,7),(6,8),(7,8)\}$$

Note how we don't add pairs corresponding to two field equations.

Now we begin processing the list L, and we choose to start with those involving the field equations. We can immediately discard $(2,5),(4,5),(1,6),(4,6),(3,7),(4,7),(2,8)$ and $(3,8)$; they correspond to pairs of field equations and polynomials, where the field equation variable does not divide the leading term of the polynomial, in accordance to Theorem 4.3. The other S-polynomials corresponding to field equations are calculated using as $(x_i + 1)p_{j,2}$, as described in Theorem 4.3. We of course use the modified multiplication when evaluating expressions. As long as we only get zeros, we will keep calculating.

$$S_{1,5} = (x_1 + 1)(x_3 + 1) = x_1x_3 + x_1 + x_3 + 1 \xrightarrow{p_5} 0$$
$$S_{3,5} = (x_3 + 1)(x_1 + 1) = x_1x_3 + x_1 + x_3 + 1 \xrightarrow{p_5} 0$$
$$S_{2,6} = (x_2 + 1)(x_3 + 1) = x_2x_3 + x_2 + x_3 + 1 \xrightarrow{p_6} 0$$
$$S_{3,6} = (x_3 + 1)(x_2 + 1) = x_2x_3 + x_2 + x_3 + 1 \xrightarrow{p_6} 0$$
$$S_{1,7} = (x_1 + 1) \cdot 0 = 0$$
$$S_{2,7} = (x_2 + 1)x_1 = x_1x_2 + x_1 \xrightarrow{p_6} 0$$
$$S_{1,8} = (x_2 + 1) \cdot 0 = 0$$
$$S_{4,8} = (x_4 + 1) \cdot 0 = 0$$

These S-polynomials are reduced to zero[6], so we do not have to add them. We now have $L = \{(5,6),(5,7),(5,8),(6,7),(6,8),(7,8)\}$ and $G = \{x_1^2 + x_1, x_2^2 + x_2, x_3^2 + x_3, x_4^2 + x_4, x_1x_3 + x_1 + x_3 + 1, x_2x_3 + x_2 + x_3 + 1, x_1x_2 + x_1, x_1x_4\}$.

Of the remaining pairs in L, we can immediately discard $(6,8)$ due to the prime criteria (Theorem 4.5).

We next pick pairs from L, calculate the S-polynomial, and attempt to reduce them

[6]We could have used the results from Section 4.4.4 for these calculations, but the improvement would have been negligible.

with p_5, p_6, p_7 and p_8[7].

$$\begin{aligned}
S_{5,6} =& x_2(x_1x_3 + x_1 + x_3 + 1) - x_1(x_2x_3 + x_2 + x_3 + 1) = \\
=& x_2x_3 + x_1x_3 + x_2 + x_1 \xrightarrow{p_5} x_2x_3 + x_1x_3 + x_2 + x_1 - (x_1x_3 + x_1 + x_3 + 1) = \\
=& x_2x_3 + x_2 + x_3 + 1 \xrightarrow{p_6} x_2x_3 + x_2 + x_3 + 1 - (x_2x_3 + x_2 + x_3 + 1) = 0 \\
S_{5,7} =& x_2(x_1x_3 + x_1 + x_3 + 1) + x_3(x_1x_2 + x_1) = \\
=& x_1x_2 + x_2x_3 + x_1x_3 + x_2 \xrightarrow{p_5} x_1x_2 + x_2x_3 + x_1x_3 + x_2 - (x_1x_3 + x_1 + x_3 + 1) = \\
=& x_1x_2 + x_2x_3 + x_1 + x_2 + x_3 + 1 \xrightarrow{p_7} x_1x_2 + x_2x_3 + x_1 + x_2 + x_3 + 1 - (x_1x_2 + x_1) = \\
=& x_2x_3 + x_2 + x_3 + 1 \xrightarrow{p_6} x_2x_3 + x_2 + x_3 + 1 - (x_2x_3 + x_2 + x_3 + 1) = \\
=& 0 \\
S_{5,8} =& x_4(x_1x_3 + x_1 + x_3 + 1) - x_3(x_1x_4) = \\
=& x_1x_4 + x_3x_4 + x_4 \xrightarrow{p_8} x_1x_4 + x_3x_4 + x_4 - (x_1x_4) = \\
=& x_3x_4 + x_4
\end{aligned}$$

As this can not be reduced further, we call it p_8, add p_8 to G and add the corresponding pairs to L. We notice that $(6, 8)$ and $(7, 9)$ can be discarded due to the prime criterion, and continue with the other S-polynomials...

$$\begin{aligned}
S_{6,7} =& x_1(x_2x_3 + x_2 + x_3 + 1) + x_3(x_1x_2 + x_1) = \\
=& x_1x_2 + x_1 \xrightarrow{p_7} x_1x_2 + x_1 - (x_1x_2 + x_1) = \\
=& 0 \\
S_{6,9} =& x_4(x_2x_3 + x_2 + x_3 + 1) + x_2(x_3x_4 + x_4) = \\
=& x_3x_4 + x_4 \xrightarrow{p_9} x_3x_4 + x_4 - (x_3x_4 + x_4) = \\
=& 0 \\
S_{5,9} =& x_4(x_1x_3 + x_1 + x_3 + 1) + x_1(x_3x_4 + x_4) = \\
=& x_3x_4 + x_4 \xrightarrow{p_9} x_3x_4 + x_4 - (x_3x_4 + x_4) = \\
=& 0 \\
S_{7,8} =& x_4(x_1x_2 + x_1) + x_2(x_1x_4) = \\
=& x_1x_4 \xrightarrow{p_8} x_1x_4 - x_1x_4 = \\
=& 0 \\
S_{8,9} =& x_3(x_1x_4) - x_1(x_3x_4 + x_4) = \\
=& x_1x_4 \xrightarrow{p_8} x_1x_4 - x_1x_4 = \\
=& 0
\end{aligned}$$

And now that L is empty, the algorithm is finished. Since G does not contain 1, we can safely draw the conclusion that ϕ is satisfiable.

4.6 Existing Implementations

The process of using Gröbner Bases to solve the SAT-problem is relatively new, but there are already several implementations out there. Anyone interested implementing

[7]We do not consider reducing it with the field equations, as that was done automatically by the modified multiplication.

this algorithm may wish to take a look at these first.

We have already mentioned SINGULAR [7], an easy-to-use algebra program, which among other things can calculate Gröbner Bases for polynomial ideals. It has a command-line driven interface, with c++-like object oriented syntax. The program contains no algorithms specific for boolean polynomials, however.

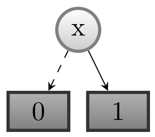
Secondly, we have Macaulay2 [8], a similar algebra program. This program does feature algorithms specialized for finding Gröbner bases for ideals of boolean polynomials, available in the package **BooleanGB** [11]. The algorithms used in this package are based on the same ideas found in this paper, but does not seem very well implemented - a few bugs has been found[8].

We also recommend PolyBoRi [9] (which stands for POLYnomials over BOolean RIngs), a c++-library completely devoted to boolean polynomials. The package features several algorithms, including efficient versions of the Buchberger algorithm. Some experiments show that PolyBoRi without a doubt are the best of the alternatives presented here. PolyBoRi is also included with most recent Sage installations and can be used with Python. The techniques used by the package, however, are not based on the same idea as in this paper, where we represent a polynomial with its coefficients. Rather, PolyBoRi uses a data structure called Binary Decision Diagrams, which will be explained in a later section. Many of the improvements discussed in this paper are not applicable to BDD's, sadly.

Finally, there is X-Solve [12], an SAT-solver which uses precisely the methods discussed in this paper. It also takes into account the hybrid method discussed in Section 5.2.

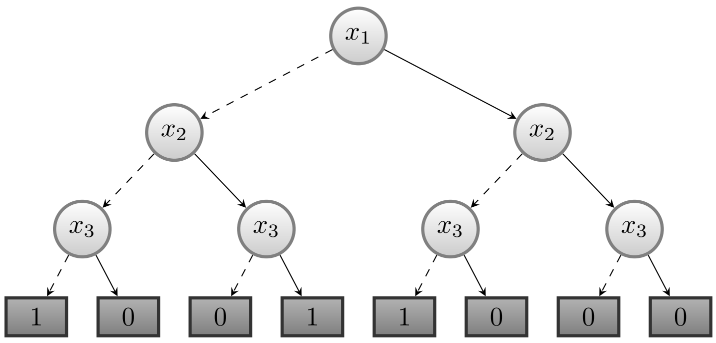
4.6.1 Binary Decision Diagrams

A boolean polynomial can be represented, not only by coefficients and terms as in this paper, but rather by what result you get when evaluating the polynomial at different points. Say for example that we had the polynomial $f = x$. There is one variable in f, which we can assign either the value 1 or 0. This choice can be used to represent the polynomial, and we can visualize it with a graph, such as this:



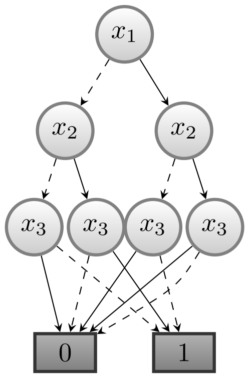
Standing at the node x and following the dashed line means that we assign the variable x the value 0, and following the whole line means that we it the value 1. If we have more variables, we simply add more nodes to the diagram: for example, this is how we represent the polynomial $x_1x_2 + x_2x_3 + x_3 + 1$:

[8]Specifically, it sometimes finds that $\{1\}$ is a Gröbner basis for an ideal I when the standard Macaulay2 functions claim that $1 \notin I$ - clearly not true.

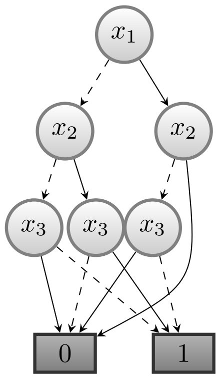


Feel free to verify that each point leads to the correct value.

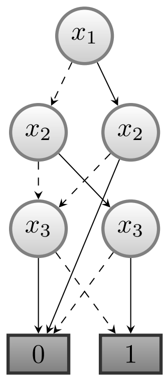
Of course, the tree doesn't have to be so large - why not just have one 1-node and one 0-node?



But some nodes are still superfluous: note the rightmost x_3-node - both its outgoing edges point to the 0-node. It doesn't matter what value we assign to x_3 at that stage, why represent the choice when it has no effect? Let's just allow the whole edge from the x_2-node to point at the 0-node immediately.



There are still more improvements available - note how both the rightmost and leftmost x_3 represent the same choice: set $x_3 = 1$ and we get 0, set $x_3 = 0$ and we get 1. The subtrees of the node are the same. Why represent the same choice twice? Let's just have one node representing both the choices.



This type of graph is called a Binary Decision Diagram, and is an alternative way of representing boolean polynomials. There are algorithms for transforming a polynomial to a diagram, adding or multiplying two diagrams, and so on [13][9]. Implementing the algorithm with this representation can be preferable to representing the coefficients.

4.6.2 Experiments

We performed a rudimentary experiment with SINGULAR, Macuaulay2 with and without the package BooleanGB, PolyBoRi, X-solve and Minisat, one of the best open source SAT-solvers available [10]. Note that Minisat does not use boolean polynomials or Gröbner bases, but is included in the experiment for comparison to conventional methods of SAT-solving.

In Singular, we measured the time it took for the program to calculate the minimal reduced Gröbner basis for the ideal generated by the boolean polynomials and the field equations. For this SINGULAR uses the regular Buchberger algorithm without any improvements discussed in this chapter, so we are not expecting a very good performance from this approach.

In Macaulay2, we first used the same method as with Singular, and expect similar results. Then we removed the field equations and used the package BooleanGB and the method gbBoolean. We expect better results using this package.

For PolyBoRi, we used the program Sage, imported the PolyBoRi commands, and let it calculate the minimal reduced Gröbner basis for the ideal generated by the boolean polynomials. As PolyBoRi is specialized in boolean polynomials, there is no need to manually add the field equations, and the Buchberger algorithm is specialized for boolean polynomials: a fairly good performance is expected.

In X-Solve and Minisat, we simply ran the program with the formulas.

The hardware used in the experiment is substandard; users with better equipment will probably see much better results. However, the experiment should still give a rough indication of the relative merits of the implementations.

We started with six formulas of various sizes[9]...

[9]The formulas were taken from http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html and we thank professor Michael Brickenstein for providing a script for converting the data to polynomials.

	Variables	Clauses	Satisfiable?
f_1	20	94	Yes
f_2	15	15	Yes
f_3	45	178	Yes
f_4	50	100	No
f_5	75	325	No
f_6	75	325	Yes

The first column shows the number of variables, the second the number of clauses, and the third whether or not the formula is satisfiable. The formulas f_1, f_4, f_5, f_6 only or mainly contain clauses of size 3, meaning that the ideal generators will have fewer terms. Meanwhile f_2, f_3 have larger formulas, corresponding to larger polynomials.

Here are the resulting times. For convenience we only let the programs run 4 hours at most.

	Singular	Macaulay2	Maulay2 with Package	PolyBoRi	X-Solve	Minisat
f_1	<1 sec	<1 sec	2,5 sec	<1 sec	<1 sec	<1 sec
f_2	<1 sec	16 sec	3 sec	<1 sec	<1 sec	<1 sec
f_3	>4 hours	>4 hours	>4 hours	12 minutes	1 sec	<1 sec
f_4	400 sec	3 sec	9 sec	<1 sec	<1 sec	<1 sec
f_5	>4 hours	>4 hours	Would not run	20 minutes	71 sec	<1 sec
f_6	>4 hours	>4 hours	Would not run	1 hour	39 sec	<1 sec

The results are the expected ones. Singular can not keep up for long before succumbing, having trouble at already at 50 variables and not even managing the larger polynomials with 45 variables.

Macaulay2 has the interesting situation of not being made much more efficient by its package: it does seem to help when dealing with larger polynomials, but actually increases the running time when the polynomials have small size. The package also can not handle more than 64 variables, severely reducing the usefulness of it.

PolyBoRi, one of the most efficient implementations for boolean polynomials available (as far as the author is aware), keeps up for much longer. X-solve, specifically designed for SAT-solving, is the fastest implementations using boolean polynomials, but is still far behind the award winning Minisat.

A word of caution: it is easy to draw the conclusion that the hybrid method used by X-solve and the BDD-data structure used by PolyBoRi would be the best implementation of boolean polynomials, but keep in mind that there are many contributing factors in these types of tests. It could very well be that PolyBoRi is simply more well-written in general and that the Macaulay2 package is in the early stages of development. That does not mean that BDD's necessarily is the better implementation technique, or that the advantage of the hybrid methods is as useful as the experiment implies.

5 Further Development and Conclusion

5.1 Finding solutions

Say that we have a polynomial ideal I containing the field equations, and that $V(I)$ only contained one point. What does that tell us about the Gröbner basis?

Say that the point in $V(I)$ was $(a_1, a_2, \ldots, a_n)$. We can then create another ideal $J\langle x_1 - a_1, x_2 - a_1, \ldots, x_n - a_n\rangle$. Then J is also solved by, and only by, $(a_1, a_2, \ldots, a_n)$, so $V(I) = V(J)$. The two ideals are also both radical, meaning that $I = J$ by Theorem 4.10, and also that their reduced minimal Gröbner bases are the same.

But look at the generators of J - they already form a Gröbner basis, which even happens to be minimal and reduced! So that is the unique minimal and reduced Gröbner basis we will get by applying the Buchberger algorithm to I! Clearly, when the variety only contains one point, the reduced minimal Gröbner basis will perfectly define that point.

Similarly, say that there are two points in the variety,
$(a_1, a_2, \ldots, a_{k-1}, 1, 0, a_{k+2}, \ldots, a_n)$ and $(a_1, a_2, \ldots, a_{k-1}, 0, 1, a_{k+2}, \ldots, a_n)$. The polynomials describing those two points would be $x_1 - a_1, x_2 - a_2, \ldots, x_{k-1} - a_{k-1}, x_k + x_{k+1} + 1, x_{k+1}^2 + x_{k+1}, x_{k+2} - a_{k+2}, \ldots, x_n - a_n$, and they happen to form a minimal reduced basis for the ideal.

The application is obvious - given a Gröbner basis, it is possible to find the points in the variety, and hence the corresponding solution to the boolean formula. As the variety grows larger, the polynomials become more and more complicated, but it does create a possibility that this method can not only be used to decide solvability, but also the actual solutions.

One problem with this approach is that it is much easier to find points in the variety this way if the Gröbner basis uses lex as its monomial ordering. However, we will want to use the monomial ordering degrevlex when performing the Buchberger algorithm, as it reduces running time, but that gives a base from which it is difficult to find solutions.

5.2 Hybrid strategies

One common strategy used by conventional SAT-solvers is to strategically attempt to assign values to the variables, and thereby reducing the size of the formula, making solutions easier to find. If no solutions exist for that particular variable assignment, the program then goes back and tries to assign other values to the variables, like a slightly more sophisticated version of the naive approach. Can a similar method be used in conjunction with our algebraic method?

It seems so: we can simulate assigning the value $\perp$ to the variable ψ_i by adding the equation $x_i - 1$ to our basis during the Buchberger algorithm. At the same time, we also save a copy of the state of the algorithm at that time. We then continue the algorithm: if the basis is found to not contain one, we can draw the conclusion that a solution exists, and if a one is found, we go back to that previous state where we assigned the value, replace the polynomial $x_i - 1$ with x_i, and continue as usual.

That way we cover all possible cases, since any potential solutions must have either that $x_i = 1$ or $x_i = 0$. This type of strategic assignment may reduce running time.

Of course, if a solution is found this way, then we don't get a perfect Gröbner basis: we get the Gröbner basis for the ideal, with the additional polynomials $x_i - 1$ or x_i added. This may make the technique less useful, depending on the situation.

The fact that SAT-solving strategies can be used to calculate Gröbner bases is also theoretically interesting - can the techniques be used to calculate bases in more general cases? Maybe other SAT-solving techniques have corresponding applications for the Buchberger algorithm? Maybe it is possible to view SAT-solving as a special case of the Buchberger Algorithm, and find techniques used in algebra to solve the SAT-problem instead?

5.3 Generalizations

The method of deciding solvability of a set polynomials using Gröbner bases can be applied on much broader subjects than just the SAT-problem. We could for example generalize it to multivalued logics, that is, boolean algebras where the variables can take on more than two values.

One such boolean algebra is Lukasiewicz's modal logic [2], where variables can be assigned the values $\top$, $\bot$ and *Unknown*, where *Unknown* represents a variable that is either $\top$ or $\bot$, but we don't know which. For example, the formulas $Unknown \land \top$ evaluates to $\top$, as the formula is evaluated to true whether or not *Unknown* represents $\top$ or $\bot$. Similarly, we get that $Unknown \lor \bot = Unknown$, as the evaluation depends entirely on the *Unknown* variable.

We can represent these types of formulas algebraically as follows: let the value $\bot$ be associated with 0, $\top$ with 1 and *Unknown* with 2, in the ring $\mathbb{Z}_3$. We can then transform the $\neg$, $\lor$ and $\land$ connectives as follows:

- $T(\neg\psi) = 2x + 1$. This gives us $T(\neg\bot) = 1$, $T(\neg\top) = 0$, $T(\neg Unknown) = 2$, as expected.
- $T(\psi_1 \land \psi_2) = x_1^2x_2^2 + 2x_1^2x_2 + 2x_1x_2^2 + 2x_1x_2$
- $T(\psi_1 \lor \psi_2) = 2x_1^2x_2^2 + x_1^2x_2 + x_1x_2^2 + x_1x_2 + x_1 + x_2$

Once a boolean formula has been transformed like this, it can be solved by calculating the Gröbner basis of the ideal generated by the polynomial, as in the Basic Algorithm. Note though that since truth is represented by 1 in this transformation, we must subtract 1 from each formula, as the method finds zeros. If the Gröbner basis contains no constant polynomials, the original boolean formula is solvable.

But the method works well even for problems not based on boolean formulas: many other problems can be described by algebraic representations, and can then be solved similarly.

The first step is to find some algebraic representation of the problem, where solutions to the problem perfectly correspond to solutions to a system of polynomials. Generate an ideal I from these polynomials. If needed, add equations to remove extraneous solutions: For example, if only integer solutions are interesting, consider adding the field equations to the ideal. In case only the solutions $x = a_i$ are interesting, the equation $(x - a_1)(x - a_2)\dots(x - a_n)$ can be added to the ideal. Once you have the ideal, use the Buchberger algorithm to find a Gröbner basis for it. Don't be afraid to modify the Buchberger algorithm - in fact, it is probably necessary to get an algorithm with acceptable efficiency. Once a Gröbner basis has been obtained, check if it contains the element one. If it does, then the original problem is unsolvable. If not, the original problem is solvable.

5.4 Conclusion

It will come as no great surprise that this algorithm will not finish in polynomial time. The time consuming part of this algorithm, the Buchberger algorithm, can actually take up to double-exponential time in the worst case, severely reducing the usefulness of this approach.

This algebraic approach does have some advantages however. Many modern SAT-solvers work by guessing solutions; very cleverly, one might add, eliminating unnecessary guesses and trying the most promising ones first. But this guessing strategy means that they have a clear advantage if a solution exists: if no solutions exist, if the formula is unsatisfiable, they tend to take slower, since they can't get "lucky" and find a solution early.

The algebraic algorithm, however, works by searching for the 1 in the Gröbner basis. It can also get lucky, by finding the 1 early. But the 1 is only in the basis if the formula is unsatisfiable! This algebraic approach should therefor tend to go faster, if the formula is unsatisfiable! The two approaches, searching for a solution and searching the Gröbner basis, complement each other in this way.

This can be illustrated by the experiment in Section 4.6.2. The formulas f_5 and f_6 contained the same number of variables and clauses, but f_5 only took a third of the time to solve by PolyBoRi. This is likely related to the fact that f_5 was unsatisfiable, and f_6 was satisfiable. Note that X-solve, using the hybrid method, handled f_6 faster than f_5 - the hybrid method works by finding solutions, just like traditional SAT-solvers.

Of course, this is all theoretical - no known implementation of this idea has even come close to beating the traditional SAT-solvers, whether the formulas are satisfiable or not. Further development is needed before this idea can be effectively used in practice.

References

[1] R. Fröberg: An Introduction to Gröbner Bases (1997), John Wiley & Sons.

[2] J. Chazarin, A. Riscos, J. A. Alonso, E. Briales: Multi-valued Logic and Gröbner Bases with Applications to Modal Logic, J. Symb. Comp. 11 (1991).

[3] R. Gebauer, H.M. Möller: On an Installation of Buchberger's Algorithm, J. Symb. Comp. 6 (1988).

[4] Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal, J. Symb. Comp. 41 (2006).

[5] M. Stone: The theory of representation for Boolean algebras, Trans. AMS 40 (1936).

[6] Bruno Buchberger: A criterion for detecting unnecessary reductions in the construction of Groebner bases, Springer L.N. in Comp. Sci. 72 (1979).

[7] W. Decker, G.-M. Greuel, G. Pfister, H. Schönemann: SINGULAR 3-1-2 — A computer algebra system for polynomial computations.
Available at `http://www.singular.uni-kl.de`.

[8] D. R. Grayson, M. E. Stillman: Macaulay2, a software system for research in algebraic geometry.
Available at `http://www.math.uiuc.edu/Macaulay2/`

[9] M. Brickenstein, A. Dreyer: PolyBoRi: A framework for Gröbner basis computations with Boolean polynomials, J. Symb. Comp. 44 (2009).

[10] N. Eén, N. Sörensson: Minisat, a minimalistic, open-source SAT solver.
Available at `http://minisat.se`

[11] F. Hinkelmann, E. Arnold: Fast Gröbner Basis Computation for Boolean Polynomials.
Preprint available at `http://arxiv.org/abs/1010.2669`

[12] D. Jacquet, S. Lundqvist, H. Rullgård: Parallellisering av en Gröbnerbasalgoritm för satisfierbarhetsproblemet. (Swedish)
Article available at `http://www2.math.su.se/~samuel/booleangb/rapport.ps`
Code available at `http://www2.math.su.se/~samuel/booleangb/X_solve_v2.tar.gz`

[13] H. R. Andersen: An Introduction to Binary Decision Diagrams.
Available at `http://www.itu.dk/people/hra`

[14] J. A. Beachy, W. D. Blair: Abstract Algebra (2006), Waveland Pr Inc.

[15] T. Yato: Complexity and Completeness of Finding another solution and its application to puzzles (2003).

[16] M. T. Goodrich, R. Tamassia: Algorithm Design (2002), John Wiley & Sons.